VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Simulátor $I^2C$ obvodu s grafickým rozhraním

# $I^2C$ Device Simulator with GUI

2014

Patrik Kortiš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Patrik Kortiš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Simulátor I2C obvodu s grafickým rozhraním
I2C Device Simulator with GUI

Zásady pro vypracování:

Navrhně programový simulátor obvodu I2C (PCF8574 a A/D převodníku). Vytvořený program bude nahrazovat fyzickou součástku programově a musí splňovat všechny standardy pro komunikaci přes rozhraní I2C.

1. Seznamte se s komunikačním rozhraním I2C.
2. Navrhněte stavový automat simulující chování podřízeného I2C zařízení.
3. Programově realizujte navržený automat.
4. Pomocí navrženého programového rozhraní realizujte simulaci dvou vybraných zařízení.
5. Navhněte vybraným součástkám GUI pro monitorování a ovládání jejich chování.
6. Zhodnoťte spolehlivost a stabilitu navrženého řešení a porovnejte chování se skutečnými součástkami.

Seznam doporučené odborné literatury:

1. I2C Specifikace: http://www.nxp.com/documents/application_note/AN10216.pdf
2. Datové listy vybraných součástek

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2014 .....

Rád bych na tomto místě poděkoval především své rodině, bez jejíž podpory by práce vznikala s daleko většími obtížemi. Veškerý čas, který jsem mohl této práci věnovat, by jinak patřil jim. Také bych rád poděkoval vedoucímu práce, Ing. Petru Olivkovi, za všechny jeho podnětné připomínky a rady.

## Abstrakt

Předmětem této práce je návrh a realizace grafického simulátoru vybraných integrovaných obvodů pro rozhraní $I^2C$. Samostatnou částí práce je realizace stavového automatu, který plní funkci zařízení typu Slave dle specifikace tohoto rozhraní a je aplikován pro simulaci funkce vybraných obvodů. Součástí práce je rovněž popis sériového rozhraní $I^2C$ a způsoby komunikace zařízení využívajících toto rozhraní pro jejich funkci.

**Klíčová slova:** I2C, automat, GUI, sběrnice, rozhraní, simulace, gtkkmm, AD převodník, expandér

## Abstract

The objective of this work is the design and implementation of graphical simulator for selected integrated circuit $I^2C$ interface. A separate section is the realization of the state machine, to fulfill the functions slave device according to specification this interface, and is applied for simulating function of selected circuits. The thesis also includes a description of the serial interface $I^2C$ and methods of communication devices using this interface for their function.

**Keywords:** I2C, machine, GUI, bus, interface, simulation, gtkmm, AD converter, expander

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| ACK | – | Acknowledge, potvrzení přijetí dat |
| AD | – | Analog To Digital, převod analogové veličiny na digitální |
| CPU | – | Central Processing Unit, hlavní procesorová jednotka, procesor |
| DA | – | Digital To Analog, převod digitální hodnoty na analogovou veličinu |
| GTKmm | – | Good To Know, knihovna pro tvorbu GUI v jazyce C++, podmnožina knihovny GTK+ |
| GUI | – | Graphics User Interface, grafické uživatelské rozhraní |
| $I^2C$ | – | Internal-Integrated-Circuit, sběrnice navržená firmou Philips |
| I/O | – | Input/Output, označení linky nebo portu s obousměrným přenosem |
| LED | – | Light Emitting Diode, světlo vyzařující dioda |
| LSB | – | Least Significant Bit, nejnižší platný bit (bit s nejnižší váhou) |
| MSB | – | Most Significant Bit, nejvyšší platný bit (bit s nejvyšší váhou) |
| NACK | – | Not Acknowledge, data nepřijata nebo další data nepožadována |
| RS-232 | – | Standard pro vzájemnou sériovou komunikaci dvou zařízení |
| SCL | – | Serial Clock Line, linka pro hodinový (synchronizační) signál |
| SDA | – | Serial Data Line, linka pro přenos datových bitů |
| TCP | – | Transmission Control Protocol, protokol pro spolehlivý obousměrný přenos dat v síti |
| USB | – | Universal Serial Bus, univerzální sériová sběrnice pro připojení periferií k počítači |

# Obsah

# Seznam tabulek

# Seznam obrázků

# Seznam výpisů zdrojového kódu

# 1 Úvod

Cílem bakalářské práce je návrh a vytvoření programového simulátoru s GUI rozhraním pro vybrané součástky (integrované obvody) používajících ke své funkci sběrnici $I^2C$. Pro simulaci byl zvolen 8bitový I/O expandér PCF8574 a jednoduchý jednokanálový 8bitový AD převodník. Simulátor se má stát pomocným nástrojem při výuce studentů v předmětech, jejichž náplní je programování komunikace se skutečnými obvody pro sběrnici $I^2C$, kde má nahradit fyzickou součástku, resp. celý funkční modul.

Simulátor má být navržen takovým způsobem, aby umožňoval jednoduché použití, přehledné zobrazení a splňoval standardy pro komunikaci obvodů na rozhraní $I^2C$. Vyvinutá aplikace má doplňovat již existující simulátor LED diod řízených pulzně šířkovou modulací [3], ze kterého koncepčně vychází.

Úvodní část práce je věnována popisu komunikačního rozhraní $I^2C$, jeho charakteristikou, popisem komunikačního protokolu, možnostmi, limity a oblastmi použití. Samostatná část práce popisuje návrh stavového automatu, který má simulovat chování obecného podřízeného (Slave) $I^2C$ zařízení. Na tuto část navazuje programová realizace stavového automatu.

V další části práce je uveden stručný popis funkcí skutečných obvodů vybraných pro simulaci.

Na předchozí části navazuje rozsáhlejší kapitola popisující návrh grafického rozhraní. Jedním z hlavních požadavků bylo navrhnout takové grafické rozhraní, které by poskytovalo jednoduché ovládání a svými možnostmi pokud možno funkčně odpovídalo skutečným součástkám. Při tvorbě grafického rozhraní bude využito podpůrných nástrojů (toolkitů), které usnadňují jejich implementaci v prostředí operačního systému Windows.

Nedílnou součástí práce bude také popis funkce klientské části řešení, tj. části jejímž úkolem je plnit roli řídícího obvodu (zařízení Master), který řídí činnost na sběrnici $I^2C$.

V závěru práce je provedeno zhodnocení navrženého řešení, jeho spolehlivosti a stability, včetně porovnání s funkcemi skutečných součástek. V této části bude rovněž zmíněna možnost přenosu řešení do prostředí operačního systému Linux.

# 2 Komunikační rozhraní I$^2$C

## 2.1 Stručný popis a historie

Rozhraní I$^2$C (čteme jako *I dva C* nebo *IIC*) je typ sériové sběrnice typu multi-master, původně navržený a vyvinutý v roce 1982 firmou Philips, dnes NXP Semiconductors. Tento typ sběrnice byl navržen k propojení malého počtu integrovaných obvodů (resp. funkčních celků) umístěných na jedné desce (plošném spoji) či v jednom zařízení pomocí minimálního počtu vodičů (dvou a země). Sběrnice I$^2$C byla primárně určena pro zařízení spotřební elektroniky, např. televizory, radiové přijímače apod.

Firmou Philips byla stanovena specifikace umožňujících jednoduché použití této sběrnice a garantující kompatibilitu při propojení integrovaných obvodů různých výrobců. Specifikace neobsahuje způsob a požadavky na fyzického provedení sběrnice, tedy nestanovuje druh kabeláže, konektorů apod. Často je sběrnice vytvořena pouze ve formě vodivých spojů na deskách s plošnými spoji.

Sběrnice od roku 1982 prošla dalším zdokonalováním (revizemi) a to především navyšováním rychlosti, kterou mohou zařízení připojená na sběrnici mezi sebou komunikovat. V roce 1992 byla navýšena komunikační rychlost z původních 100 kHz na 400 kHz (Fast Mode) a byly rozšířeny adresní možnosti z původních 7 bitů na 10 bitů. Další vývoj sběrnice je uveden v tabulce 1. V roce 2012 byla vydána specifikace zatím poslední verze.

| Rok | Verze | Maximální frekvence na lince SCL, změny |
|---|---|---|
| 1982 | – | Frekvence 100 kHz |
| 1992 | verze 1 | Přidán mód *Fast-mode* 400 kHz a 10bitové adresování. První standardizovaná verze. |
| 1998 | verze 2 | Přidán *High-speed mode* 3,4 MHz s definici požadavků pro úsporu energie |
| 2007 | verze 3 | Přidán *Fast-mode plus* 1 MHz |
| 2012 | verze 4 | Přidán *Ultra Fast-mode* 5 MHz pro nové linky USCL a USDA |
| 2012 | verze 5 | Korekce chyb ve verzi 4 |

Tabulka 1: Vývoj rozhraní I$^2$C

Maximální dovolená kapacita na linkách sběrnice vůči společnému vodiči (zemi) byla stanovena na 400 pF s ohledem na strmost hran signálů při stanovené maximální rychlosti přenosu 100 kbps.

Sběrnice I$^2$C je i přes své relativní stáří v dnešní době mezi vývojáři stále velmi populární, existuje velké množství obvodů rozličných funkcí od mnoha výrobců. Na sběrnici I$^2$C může být připojen jakýkoliv obvod, který ovládá protokol sběrnice. Mimo integrovaných obvodů RAM, E$^2$PROM, obvodů pro rozšíření portů, A/D a D/A převodníků, teplotních senzorů a obvodů reálného času existuje celá řada dalších speciálních integrovaných obvodů, jako například budiče displejů nebo integrovaných obvodů pro video a

audio techniku. Sběrnice se také uplatňuje jako servisní port pro monitorování a odstraňování závad u rozličných zařízení spotřební a jiné elektroniky.

## 2.2 Terminologie

Tabulka 2 popisuje význam často užívaných pojmů při popisu rozhraní $I^2C$.

| **Pojem** | **Popis** |
|---|---|
| Master | Zařízení (obvod), které zahajuje a ukončuje přenos. Master také generuje hodinový takt na lince SCL. |
| Slave | Zařízení (obvod) adresované Masterem. |
| Vysílač | Zařízení (obvod) vystavující data na sběrnici. |
| Přijímač | Zařízení (obvod), které data ze sběrnice čte. |
| Multi-Master | Existence více než jednoho Master zařízení na stejné sběrnici ve stejný čas bez kolizí a ztráty dat. |
| Arbitrace | Postup, který umožňuje pouze jednomu zařízení Master převzít kontrolu nad sběrnicí. |
| Synchronizace | Postup, který umožňuje synchronizaci hodinových signálů ze dvou nebo více zařízení. |
| SCL | Linka s hodinovým signálem (Serial CLock). |
| SDA | Linka s datovým signálem (Serial DAta). |

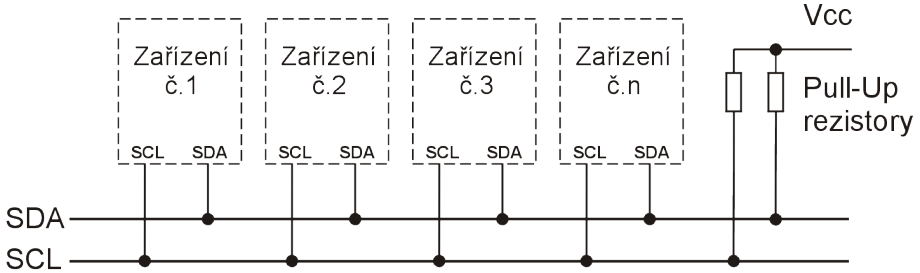
Tabulka 2: Často užívané pojmy

## 2.3 Základní vlastnosti sběrnice

- Potřebuje pouze dvě linky: datovou (SDA) a hodinovou (SCL).
- Každé zařízení připojené na sběrnici je adresovatelné prostřednictvím své jedinečné adresy.
- Zařízení Master může pracovat jako vysílač nebo jako přijímač.
- Sériový přenos, s délkou slova 8 bitů, obousměrný datový přenos probíhající maximální rychlostí 100 kbit/s ve standardním módu nebo až 400 kbit/s v módu Fast, příp. až 3,4 Mbit/s v módu High–speed.
- Na stejnou sběrnici může být připojen větší počet integrovaných obvodů (je limitován pouze maximální kapacitou linek do 400 pF).

## 2.4 Elektrické propojení

Jak už bylo uvedeno výše, jsou zařízení na sběrnici I²C propojena prostřednictvím dvou linek označovaných jako SDA pro data a SCL pro hodinový signál, viz obr. 1. Obě linky pracují nejčastěji s napěťovými úrovněmi technologie TTL[1], nicméně existují i variace pracující na nižších napěťových úrovních, např. 3,3 V.




Obrázek 1: Propojení zařízení na sběrnici I²C

Prvotní specifikace umožňovala adresovat na sběrnici zařízení v rozsahu 7 bitů, tj. až 128 různých zařízení (prakticky je však toto číslo menší o několik rezervovaných adres). Po rozšíření na 10 adresních bitů se adresní možnosti zvětšily až na 1008 zařízení.

K implementaci sběrnice v koncovém zařízení je potřeba ještě dvou zdvihacích (pull-up) rezistorů, připojených k oběma linkám vůči kladnému napájecímu napětí. Vzhledem k tomu, že linky SCL a SDA jsou ve vnitřních strukturách obvodů implementovány jako tranzistory s otevřenými kolektory (viz obr. 2), udržují připojené rezistory obě linky ve stavu logické jedničky. V době, kdy jsou obě linky ve stavu logické jedničky, je stav na sběrnici považován za klidový stav a všechna zařízení na sběrnici jsou nečinná (mohou být klidně i odpojena od sběrnice). Sběrnice může v tomto stavu zůstat po libovolně dlouhou dobu, některé obvody, především jednočipové mikroprocesory, dokonce přecházejí do režimu spánku s tím, že v případě potřeby komunikace se automaticky aktivují. Vnitřní implementací datových linek pomocí tranzistorů s otevřenými kolektory je zajištěno, že kterýkoli obvod připojený na sběrnici může uvést příslušnou linku do stavu logické nuly uzavřením příslušného tranzistoru, aniž by tím způsoboval nepřípustné stavy na výstupech jiných obvodů připojených na sběrnici.

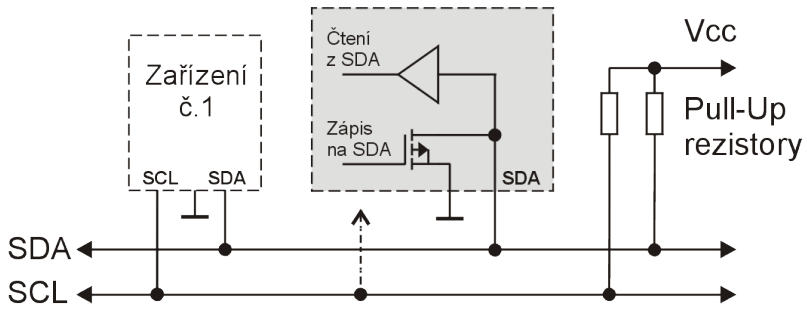
Data (a adresy) se přenášejí s náběžnou hranou hodinových impulsů. Obě linky je možno používat jako obousměrné, v daném okamžiku ale pouze jedním směrem.

## 2.5 Datový přenos a formát rámce

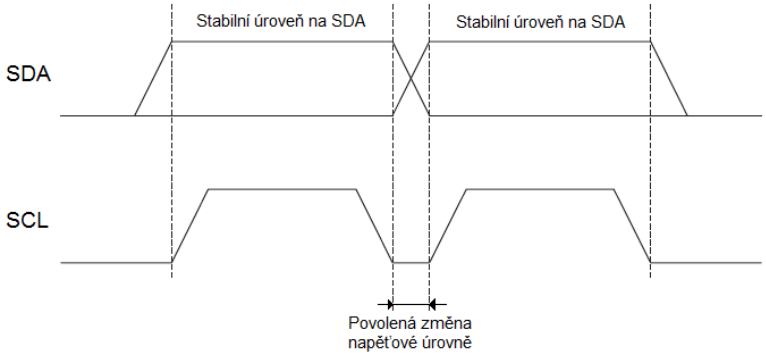
### 2.5.1 Přenos bitů

Každý datový bit je na sběrnici doprovázen pulsem na lince SCL. Napěťová úroveň na datové lince SDA musí být v době vysoké úrovně na lince SCL stabilní (viz obr. 3). Jediné výjimky z tohoto pravidla jsou přípustné při generování podmínky START a STOP.

[1] Transistor-Transistor-Logic; Obvody technologie TTL používají napájecí napětí 4,5 až 5,5 V, za logickou jedničku je považováno napětí nad 2 V, za logickou nulu napětí do 0,8 V.



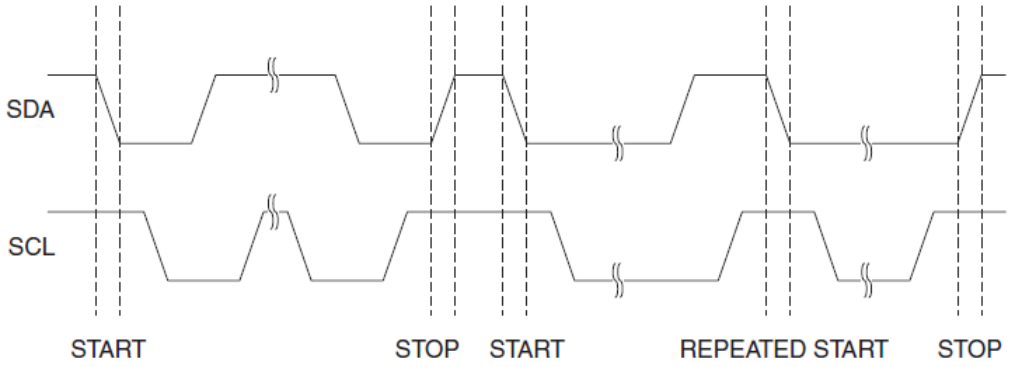

Obrázek 2: Zjednodušené vnitřní zapojení linek SDA a SCL




Obrázek 3: Stabilita napěťové úrovně linky SDA v době vysoké úrovně na SCL

### 2.5.2 Podmínky START a STOP

Zařízení Master zahajuje a ukončuje každý přenos na sběrnici. Přenos je zahájen v okamžiku, kdy zařízení Master vyšle (nastaví) na sběrnici podmínku START a je ukončen v okamžiku, kdy zařízení Master vyšle na sběrnici podmínku STOP. Mezi oběma podmínkami je sběrnice považována za obsazenou a pokud se v této době pokusí jiné zařízení Master na stejné sběrnici zahájit přenos, nepodaří se mu to. Speciálním případem je však situace, kdy je v době obsazené sběrnice vyslána zařízením Master nová podmínka START. Taková situace je označována jako vyslání podmínky REPEATED START (opakovaný start) a je používána v případě, kdy si zařízení Master přeje iniciovat nový přenos aniž by ztratil kontrolu nad sběrnicí. Po vyslání podmínky REPEATED START je sběrnice opět považována, shodně jako u podmínky START, za obsazenou až do vyslání podmínky STOP. Mezi oběma podmínkami START a STOP může být vyslán neomezený počet podmínek REPEATED START. Jak je zobrazeno na obr. 4, podmínky START a STOP jsou vyslány změnou úrovně na SDA lince v době, kdy je linka SCL ve stavu logické jedničky.




Obrázek 4: Vyslání podmínky START, REPEATED START a STOP

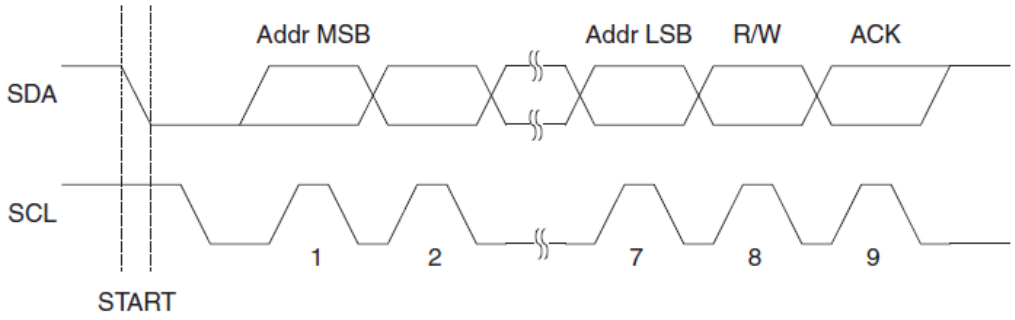
### 2.5.3 Formát adresního rámce

Všechny datové rámce vyslané na sběrnici obsahující adresu, jsou dlouhé 9 bitů. Sedm bitů obsahuje adresu zařízení, jeden bit je příznakem pro směr přenosu (READ/WRITE bit) a jeden bit slouží k potvrzení přenosu cílovým zařízením (tzv. Acknowledge bit). Pokud je READ/WRITE bit nastaven, bude docházet k operaci čtení, v opačném případě bude docházet k zápisu. Pokud je vyslaná adresa Slave zařízením přijata, pak toto zařízení potvrdí svou přítomnost a připravenost stažením úrovně na SDA lince v devátém SCL (ACK) cyklu. Pokud je adresované Slave zařízení zaneprázdněno nebo z nějakých jiných důvodů nemůže vyhovět požadavku zařízení Master, ponechá úroveň na SDA lince v devátém SCL cyklu na vysoké úrovni. Zařízení Master pak může reagovat buď ukončením přenosu vysláním podmínky STOP nebo vysláním podmínky REPEATED START k iniciaci nového přenosu.

Při přenosu adresy je nejprve vysílán nejvyšší platný bit (MSB). Adresa `0000 000` je rezervována pro speciální účely, pro tzv. *General Call* (obecné volání). Jedná se o adresu, kterou zařízení Master oslovuje všechna zařízení připojená na sběrnici (lze přirovnat k broadcastu známého z Ethernetu).

Pokud je zařízením Master vyslána na sběrnici adresa pro obecné volání, měla by všechna zařízení Slave připojená na sběrnici reagovat stažením úrovně na SDA lince v devátém cyklu hodinového signálu. Obecné volání používá zařízení Master pokud chce několika zařízením Slave předat stejnou zprávu. Pokud adresa pro obecné volání má nastaven bit READ/WRITE na hodnotu indikující zápis (logická nula), pak všechna Slave zařízení akceptující obecné volání nastaví v devátém cyklu (ACK) SDA linku na nízkou úroveň. Následující datové pakety pak budou přijímány všemi Slave zařízeními, které akceptovaly výzvu obecného volání. Je nutno poznamenat, že vyslání adresy pro obecné volání následované nastavením příznaku READ/WRITE na hodnotu indikující čtení (logická jednička) nemá smysl, neboť by způsobilo na datové lince SDA kolize vysíláním odlišných dat několika Slave zařízeními.

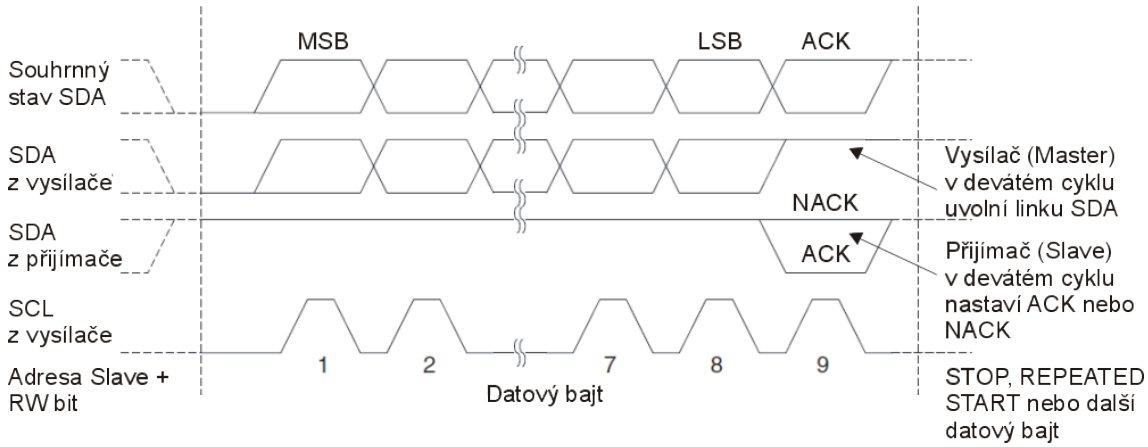
Všechny adresy formátu `1111 xxx` jsou rezervovány pro budoucí použití.




Obrázek 5: Formát rámce s adresou

### 2.5.4 Formát datového rámce

Všechny rámce s daty vysílané na sběrnici, jsou stejně jako rámce s adresou, 9 bitů dlouhé. Obsahují jeden datový bajt a jeden bit pro potvrzení přenosu přijímajícím zařízením. V průběhu datového přenosu generuje zařízení Master na lince SCL hodinový signál, podmínky START/STOP a zařízení Slave je zodpovědné za potvrzování přijetí datového bajtu. Potvrzení (ACK) je zařízením Slave signalizováno nastavením linky SDA na nízkou úroveň v devátém cyklu hodinového signálu. Pokud zařízení Slave ponechá linku SDA na vysoké úrovni, signalizuje NACK. Pokud Slave zařízení obdrželo poslední datový bajt nebo z nějakého důvodu nechce nebo nemůže přijímat další data, měl by informovat zařízení Master nastavením NACK po přijetí posledního datového bajtu. Při přenosu datového bajtu je nejprve vysílán nejvyšší platný bit (MSB).




Obrázek 6: Formát rámce s daty

### 2.5.5 Kombinace adresových a datových rámců během přenosu

Celý přenos je složen v uvedeném pořadí z podmínky START, adresy Slave zařízení včetně READ/WRITE bitu, jednoho nebo více datových paketů a STOP podmínky. Přenos, který obsahuje pouze podmínku START následovanou podmínkou STOP (tzv. prázdnou zprávu) je zakázán. Způsob implementace linky SCL (otevřený kolektor) umožňuje vzájemné řízení přenosu mezi zařízením Master a zařízením Slave. Zařízení Slave může prodloužit dobu periody hodinového signálu stažením úrovně linky SCL. Tímto způsobem může zařízení Slave ovlivňovat rychlost přenosu v případě, kdy je zařízení Master příliš rychlé oproti zařízení Slave, např. když zařízení Slave potřebuje jistý čas na zpracování přijatých dat. Prodlužování nízké úrovně na lince SCL ze strany zařízení Slave nemá žádný vliv na délku v úrovni logické jedničky, neboť řízení je v tomto případě vždy v režii zařízení Master.

Na obrázcích 7 až 10 je zobrazen typický průběh datových přenosů. Jako příklad je na obr. 7 uveden průběh zápisu dat na zařízení Slave, na obr. 8 čtení dat ze zařízení Slave, na obr. 9 kombinace zápisu dat následované čtením dat a na obr. 10 kombinace čtení dat následované zápisem dat. Poznamenejme, že mezi podmínkou START a podmínkou STOP může být vyslán neomezený počet datových paketů, vždy záleží na konkrétní implementaci aplikačního softwaru.

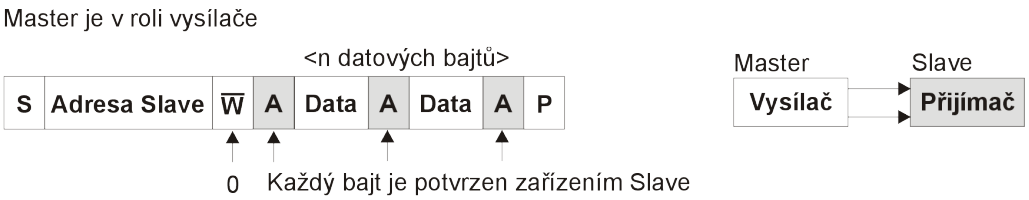
Význam použitých zkratek:

- **S (START) nebo Sr (REPEATED START)** – datový přenos začíná po vyslání START podmínky. Úroveň na SDA lince se změní z vysoké na nízkou.
- **P (STOP)** – datový přenos je ukončen vysláním podmínky STOP. V době, kdy je na lince SCL úroveň vysoká, přejde linka SDA z nízké do vysoké úrovně.
- **A (ACK)** – zařízení Master nebo Slave potvrzuje (Acknowledge) přijetí bajtu stažením úrovně na lince SDA.
- $\overline{\mathbf{A}}$ **(NACK)** – zařízení Master nebo Slave nepotvrzuje (Not Acknowledge) přijetí bajtu ponecháním vysoké úrovně na lince SDA.
- $\overline{\mathbf{W}}$ – příznak zápisu (úroveň logické nuly)
- **R** – příznak čtení (úroveň logické jedničky)

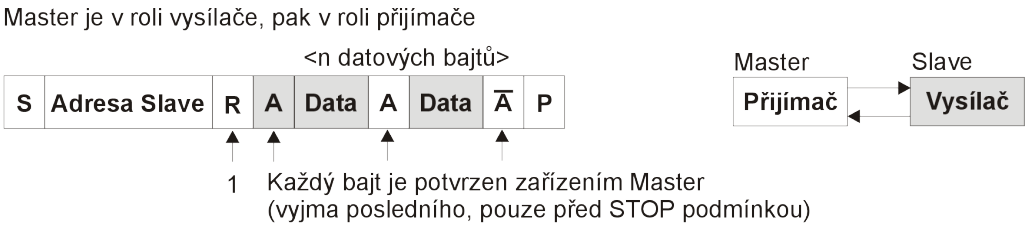
### 2.5.6 Použití desetibitové adresace

Dle prvotní specifikace rozhraní I$^2$C bylo možné na jedné sběrnici adresovat maximálně 128 zařízení. Ve skutečnosti však byl počet zařízení značně menší, neboť je nutné odečíst adresy se speciálním významem (adresa pro obecné volání, adresy s prefixem `1111 xxx`).

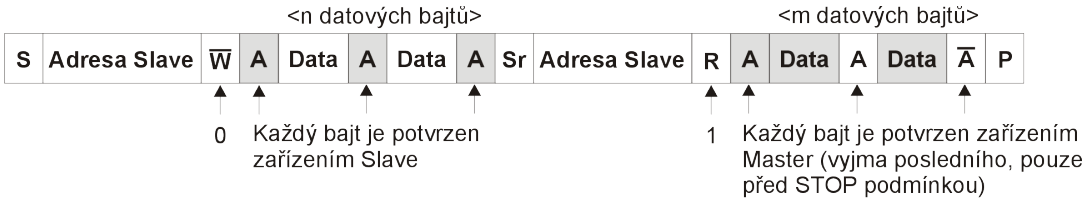
Verze 1 rozhraní I$^2$C umožnila adresaci připojených zařízení v rozsahu 10 bitů, tj. maximálně 1024 zařízení, po odečtu vyhrazených adres celkem 1008 zařízení.



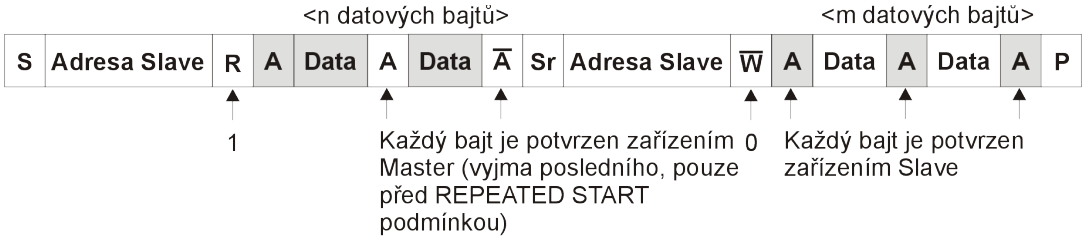

Obrázek 7: Zápis dat na zařízení Slave



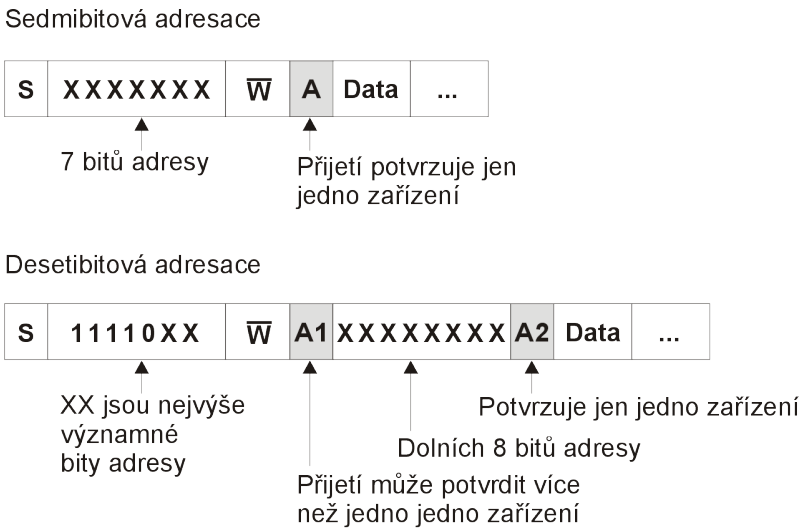

Obrázek 8: Čtení dat ze zařízení Slave




Obrázek 9: Kombinace zápisu a čtení dat na/ze zařízení Slave




Obrázek 10: Kombinace čtení a zápisu dat ze/na zařízení Slave




Obrázek 11: Rozdíly v adresaci 7 a 10bitových zařízení

Některá zařízení připojená na sběrnici I²C tedy mohou být nakonfigurována tak, že budou přijímat desetibitovou adresu. Při přenosu 10bitové adresy se využívá rozsah adres vyhrazených pro speciální účely, tj. právě adresy s prefixem `1111 xxxx`. Adresa o rozsahu 10 bitů se přenáší ve dvou bajtech a to tak, že v prvním bajtu jsou uloženy dva nejvyšší platné bity adresy a druhý bajt obsahuje dolních osm bitů adresy, viz obr. 11. První bajt adresy je ve tvaru `11110xx0`, kde xx má hodnotu desátého a devátého bitu adresy. Nejnižší platný bit (LSB) prvního bajtu adresy musí být nastaven na úroveň logické nuly, protože zařízení Slave musí přijmout i druhý bajt adresy (zařízení Slave bude očekávat další bajt). Po přenosu druhého bajtu může zařízení Master zahájit přenos dat na adresované zařízení Slave, přičemž toto zařízení potvrzuje příjem každého datového bajtu stejným způsobem jako příjem adresy, tj. v devátém cyklu hodin stažením linky SDA do nízké úrovně (ACK).

### 2.5.7 Multi-Master systémy

Specifikace rozhraní I²C umožňuje, aby na totožné sběrnici bylo více zařízení typu Master. Přítomnost více jak jednoho zařízení typu Master na stejné sběrnici přináší níže uvedené problémy.

- Prvním problémem je zamezení současného přenosu dat více zařízeními typu Master. Je přípustné, aby jen jedno zařízení Master provádělo přenos. Všechna ostatní zařízení typu Master musí přestat vysílat, když zjistí, že ztratila tzv. arbitraci.
- Druhým problémem je synchronizace hodin na lince SCL. Zařízení Master mohou na sběrnici používat odlišné frekvence hodinového signálu. K zajištění hladkého přenosu dat na sběrnici musí být cílový systém navržen tak, aby synchronizoval hodiny všech zařízení Master (tím bude usnadněn arbitrážní proces).

Vzhledem k tomu, že cílem bakalářské práce je simulace provozu na sběrnici, na níž pracuje pouze jedno zařízení typu Master (klientský software) a několik zařízení typu Slave (Simulátor), nebudeme se zabývat rozborem řešení výše uvedených problémů. Bližší informace o použití Multi-Master systémů lze získat v [1].

### 2.5.8 Přenosové rychlosti

Specifikace rozhraní I$^2$C definuje několik standardních přenosových rychlostí, které pokrývají většinu praktických aplikací. V tabulce 3 jsou uvedeny rychlosti přenosu odvozené přímo z frekvence hodinového signálu, přičemž nejsou do těchto hodnot započítány potvrzovací bity. Vzhledem k tomu, že se osm datových bitů přenáší minimálně v devíti hodinových cyklech (poslední cyklus je rezervovaný pro potvrzovací signál ACK), je reálná přenosová rychlost užitečných dat o něco nižší.

| **Přenosová rychlost** | **Označení módu** |
|---|---|
| 100 kbps | Standard mode |
| 400 kbps | Fast-Mode |
| 1 Mbps | Fast-Mode plus |
| 3,4 Mbps | High-speed mode |
| 5 Mbps | Ultra-Fast mode |

Tabulka 3: Přenosové rychlosti dle specifikace

V této souvislosti je rovněž nutno zmínit, že dle specifikace rozhraní I$^2$C jsou definovány pouze maximální přenosové rychlosti. Rychlost není nijak zdola omezena a zařízení na rozhraní mohou pracovat na libovolné frekvenci do stanoveného maxima.

# 3 Stavový automat I²C zařízení

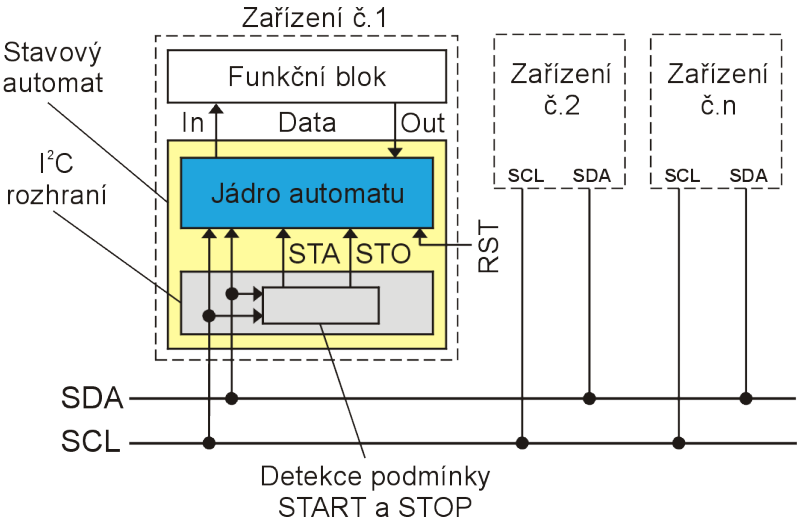
Obsahem této kapitoly je popis a rozbor funkce stavového automatu, který odpovídá specifikaci rozhraní I²C pro zařízení typu Slave.

## 3.1 Koncepce zařízení Slave

Aby bylo možné realizovat libovolné Slave zařízení pro sběrnici I²C, je nutné navrhnout autonomní modul, který bude umět ovládat signály SCL a SDA, dokáže v těchto signálech rozlišit změnu úrovní a detekovat podmínky START a STOP. Výstupem takového modulu musí být vstupně/výstupní brána o šířce jednoho bajtu. Funkci takového modulu bude zastávat stavový automat s konečným počtem stavů.

Zařízení Slave bude mít koncepci znázorněnou na obrázku 12. V této koncepci je zařízení Slave rozděleno na dvě vrstvy, které mezi sebou spolupracují. Horní vrstvu tvoří *funkční blok* pro příjem nebo vysílání dat a který realizuje příslušnou funkci zařízení (např. AD převod). Dolní vrstvu tvoří *rozhraní* mezi funkčním blokem a sběrnicí I²C. Úkolem tohoto rozhraní je zprostředkovat datovou komunikaci mezi funkčním blokem a sběrnicí.

Rozhraní mezi funkčním blokem a sběrnicí je tvořeno stavovým automatem řídícím činnost tohoto rozhraní. Stavový automat zahrnuje rozhraní I²C umožňující čtení a zápis na linky SCL a SDA. Úkolem rozhraní I²C je také detekce podmínky START a STOP.




Obrázek 12: Koncepce zařízení Slave

Při zvolené koncepci funkční blok v cílové aplikaci využívá pouze funkce pro vstup a výstup dat, které poskytuje rozhraní dolní vrstvy. Rozhraní se stavovým automatem tím zcela před funkčním blokem zakrývá detaily komunikace na sběrnici I²C.

## 3.2 Popis funkce stavového automatu

Na obrázku 13 je zobrazen stavový diagram automatu. Stavový diagram obsahuje 7 stavů pojmenovaných písmeny **A–G** a přechody mezi těmito stavy, které jsou prováděny na vzestupné hraně signálu na lince SCL generované zařízením Master. Činnost stavového automatu musí odpovídat specifikaci rozhraní. Zařízení Slave musí vždy pracovat synchronně se zařízením Master, tzn. že úroveň na lince SDA je možno měnit jen pokud je linka SCL v úrovni logické nuly.

Po inicializaci (resetu) stavového automatu přechází automat do stavu **A** (IDLE). V případě, že je automatem detekována podmínka START, automat přechází do stavu **G** (Start). Do tohoto stavu může automat přejít kdykoli je automatem detekována podmínka START, nehledě na aktuální stav automatu. Toto chování reflektuje situaci na skutečné sběrnici, na které se může kdykoliv během přenosu (asynchronně) vyskytnout podmínka START. Při normální funkci se však automat dostává do stavu **G** pouze ze stavu **A**.

Po detekování podmínky START automat přechází do stavu **B** (ACCEPT_ADDRESS), kde od zařízení Master přijímá adresu a informaci o směru přenosu. Pokud přijatá adresa neodpovídá adrese Slave zařízení, automat přechází do stavu **A**. Pokud je směr přenosu READ (bit R/$\overline{W}$ je nastaven), automat přejde do stavu **C** (LOAD_DATA); v opačném případě (bit R/$\overline{W}$ je v nízké úrovni) přejde do stavu **E** (CLEAR_DATA). Když automat přejde do stavu **C** nebo **E**, potvrzuje příjem adresy a připraví se podle směru přenosu na požadované čtení nebo zápis dat. Obě tyto operace jsou myšleny z pohledu zařízení Master. Z pohledu automatu (Slave zařízení) je operace „čtení“ interpretována tak, že zařízení Master požaduje po zařízení Slave data, které bude zasílat. Operace „zápis“ je naopak interpretována tak, že zařízení Master bude data na zařízení Slave zasílat.

Ve stavu **C** jsou data (datový bajt), která mají být zařízení Master odeslána, nejprve načtena do vnitřního buferu[2] a poté automat přechází do stavu **D** (SEND_DATA), kde jsou načtená data přenášena do zařízení Master. Pokud zařízení Master potvrdí příjem dat (datového bajtu), automat přechází zpět do stavu **C**, v opačném případě přejde do stavu **A** a čeká na příchod další podmínky START. Zařízení Master má plně v kompetenci množství přijatých dat ovládáním potvrzovacího bitu. V případě, že další data již nebude požadovat, ponechá po přijetí posledního datového bajtu linku SDA v úrovni logické jedničky (NACK) a tím způsobí, že automat po odeslání datového bajtu přejde do stavu **A**.

Ve stavu **E** automat připraví (nuluje) vnitřní bufer na příjem datového bajtu a přejde do stavu **F** (ACCEPT_DATA), kde přijímá data zasílaná zařízením Master. Po přijetí každého datového bajtu, automat zasílá signál potvrzující příjem (nastavením linky SDA na úroveň logické nuly) a vrací se zpět do stavu **E**, kde provede přípravu na příjem dalšího datového bajtu. Pokud je Slave zařízení neschopno přijmout další datový bajt, ponechá linku SDA v úrovní logické jedničky (NACK) a tím tuto skutečnost signalizuje zařízení Master a vrací se do stavu **A**. Pokud zařízení Master vyšle jinou podmínku START, automat přejde do stavu **B**. V případě, že zařízení Master vyšle podmínku STOP, automat se vrací do stavu **A** a čeká na příchod další podmínky START.

[2] Vyrovnávací paměť (anglicky *buffer*) je část paměti, která je určena pro dočasné uchování dat.

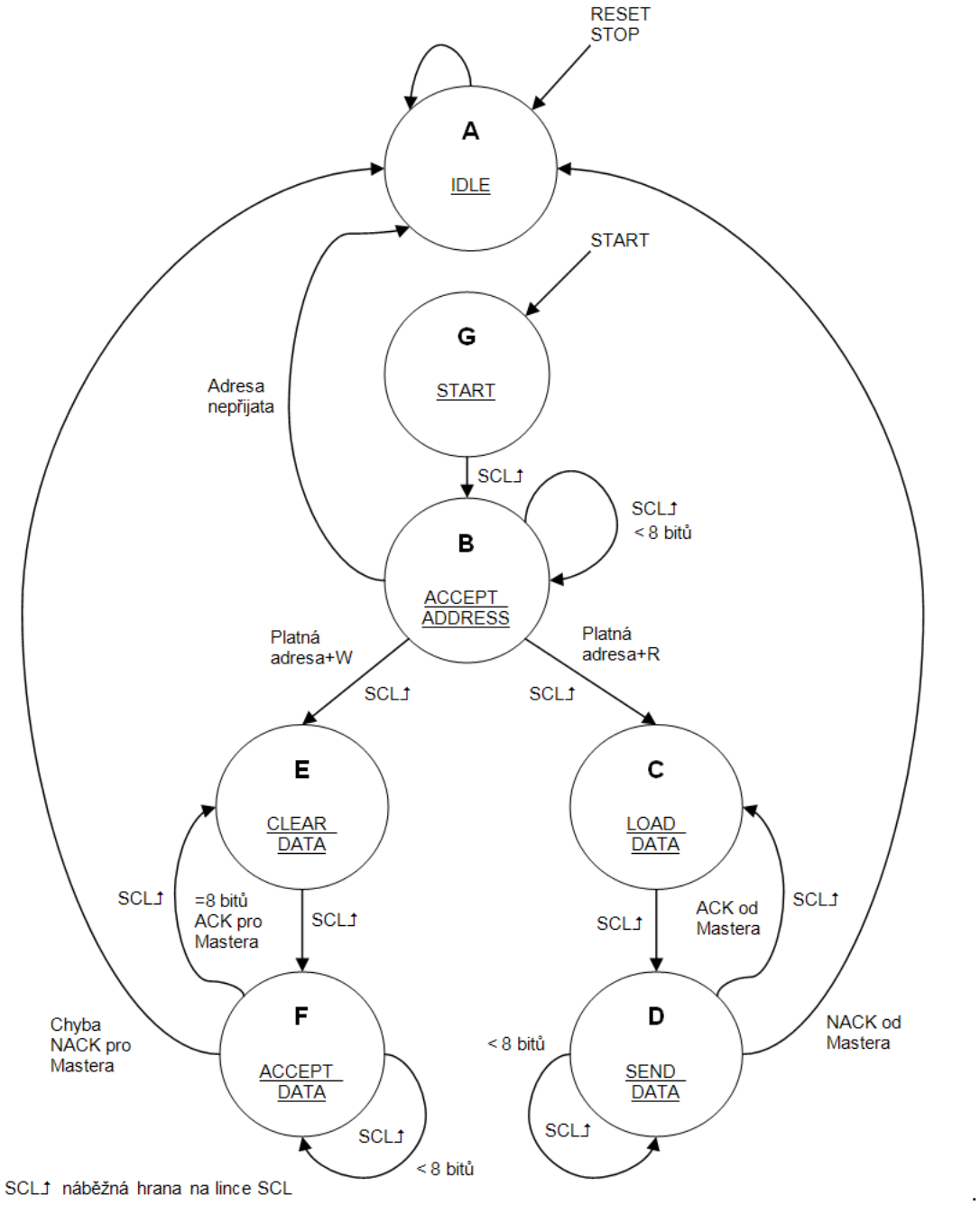
Automat ve stavech **B**, **D** a **F** setrvává po dobu osmi cyklů hodinového signálu na lince SCL, kdy s každou náběžnou hranou signálu čte stav linky SDA nebo na linku SDA zapisuje. Tento jeden cyklus odpovídá příjmu nebo zaslání jednoho bitu adresy nebo dat.

## 3.3 Operace stavového automatu

Tabulka 4 ilustruje všechny operace stavového automatu. Jak je z tabulky patrné, každý ze stavů A – F zahrnuje na svém vstupu podmínku START s následným přechodem do stavu G. Hvězdičky u některých vstupních podmínek indikují asynchronní okamžitý přechod automatu do následujícího stavu bez ohledu na hodinový signál na lince SCL, který přenos ovládá.

| **Stav** | **Název stavu** | **Vstup** | **Násl. stav** | **Akce** |
|---|---|---|---|---|
| A | IDLE | START | G | |
| B | ACCEPT_ADDRESS | $<$ 8 bitů | B | |
| | | Platná adresa + R | C | Nastavení ACK |
| | | Platná adresa + W | E | Nastavení ACK |
| | | Neplatná adresa | A | |
| | | STOP*/RESET* | A | |
| | | START* | G | |
| C | LOAD_DATA | (null) | D | Načtení dat z buferu |
| | | STOP*/RESET* | A | |
| | | START* | G | |
| D | SEND_DATA | $<$ 8 bitů | D | |
| | | ACK od Mastera | C | |
| | | NACK od Mastera | A | |
| | | STOP*/RESET* | A | |
| | | START* | G | |
| E | CLEAR_DATA | (null) | F | Vynulování buferu |
| | | STOP*/RESET* | A | |
| | | START* | G | |
| F | ACCEPT_DATA | $<$ 8 bitů | F | Uložení dat do buferu |
| | | = 8 bitů | E | Nastavení ACK |
| | | Chyba příjmu | A | Nastavení NACK |
| | | STOP*/RESET* | A | |
| | | START* | G | |
| G | START | (null) | B | |
| | | STOP*/RESET* | A | |

Tabulka 4: Operace stavového automatu




Obrázek 13: Stavový automat zařízení I²C Slave

## 3.4 Popis stavů automatu

V této části jsou shrnuty a popsány funkce jednotlivých stavů automatu.

**A (IDLE)** Stav po inicializaci nebo resetu nebo přijetí podmínky STOP. Linky SDA a SCL v úrovní HIGH, čeká se na podmínku START. Zařízení při změnách úrovně na lince SCL nemění stav.

**B (ACCEPT_ADDRESS)** Stav, ve kterém dochází ke čtení bitů adresy a směru přenosu. Stav rovněž vyhodnocuje platnost adresy a pokud přijatá adresa není platná, vrací se zařízení do stavu IDLE. Pokud je přijatá adresa platná a směr přenosu je W (Master bude zasílat data), přejde zařízení do stavu CLEAR_DATA. Pokud je směr přenosu R (Master bude číst data), přejde zařízení do stavu LOAD_DATA.

**C (LOAD_DATA)** V tomto stavu jsou připravena data zařízení Slave pro odeslání. Bufer je naplněn daty, která budou odesílána a je inicializován čítač bitů.

**D (SEND_DATA)** V tomto stavu jsou zařízením Slave odeslána data Masterovi. S každou vzestupnou hranou signálu SCL je na SDA vysláno celkem 8 bitů. V následující vzestupné hraně SCL je přečten stav linky SDA (Master signalizuje přijetí/nepřijetí dat). Pokud je linka SDA nastavena na LOW (Master potvrzuje přijetí dat $\Rightarrow$ ACK), přejde zařízení do stavu LOAD_DATA. Pokud je linka SDA nastavena na HIGH (Master nepotvrzuje či nepožaduje další data), přejde zařízení do stavu IDLE.

**E (CLEAR_DATA)** V tomto stavu se zařízení Slave připravuje pro příjem dat. Resetuje bufer a inicializuje čítač bitů.

**F (ACCEPT_DATA)** V tomto stavu jsou zařízením Slave přijímána data od zařízení Master. S každou vzestupnou hranou signálu SCL je na lince SDA přijato celkem 8 bitů. Po přijetí všech 8 bitů zařízení Slave nastaví linku SDA na LOW, čímž signalizuje zařízení Master v dalším cyklu SCL přijetí dat. Zařízení Slave poté přejde do stavu CLEAR_DATA.

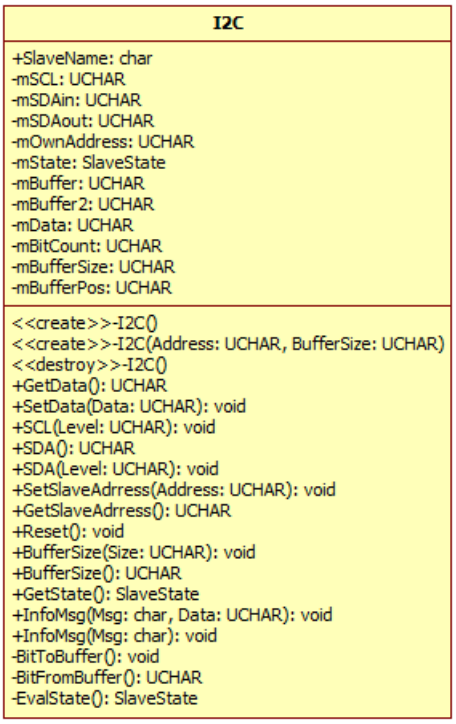
**G (START)** Stav po přijetí podmínky START. Do tohoto stavu může zařízení přejít asynchronně z kteréhokoliv jiného stavu. Přestože je zařízení ovládáno signálem SCL, výskyt podmínky START má vyšší prioritu.

# 4 Realizace stavového automatu

## 4.1 Způsob implementace

Navržený stavový automat je programově realizován jako samostatná třída s názvem I2C. Tato třída poskytuje na svém veřejném rozhraní jak stěžejní metody SCL a SDA simulující reálné linky sběrnice, metody umožňující přístup k vnitřnímu buferu GetData a SetData, tak i další pomocné metody, např. GetState vracející aktuální stav automatu. Třída I2C dále obsahuje řadu privátních atributů nutných ke správné funkci automatu, např. bufery pro data, čítač cyklu, provozní příznaky atd. Rozhraní třídy je implementováno takovým způsobem, aby se co nejvíce přibližovalo obecnému Slave zařízení.

Pomocí obecné třídy lze vytvářet v aplikaci objekty reprezentující simulované Slave zařízení, případně s využitím dědičnosti vytvářet nové třídy se specifickými vlastnosti. Na obrázku 14 je zobrazeno rozhraní třídy I2C.




Obrázek 14: Rozhraní třídy $I^2C$

Metody SCL a SDA na svém vstupu nejprve vyhodnocují typ hrany signálu (náběžná, sestupná, bez změny). Metoda SCL s každou náběžnou hranou signálu SCL vyhodnocuje nový stav automatu a provede příslušné operace odpovídající danému stavu. Metoda SDA vyhodnocuje, s ohledem na stav linky SCL, příchod podmínky START nebo STOP. V případě, že je jedna z podmínek detekována, změní stav automatu.

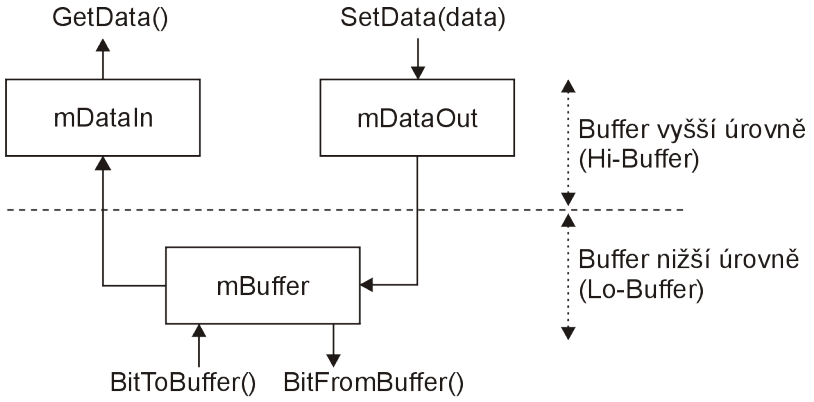
Důležitou metodou třídy I2C je metoda EvalState, která rozhoduje o novém stavu automatu. Tato metoda v souladu s funkcí stavového automatu (viz obr. 13) definuje nový stav automatu.

Vnitřní implementace metod SCL, SDA a EvalState je znázorněna na operačních diagramech uvedených v příloze A.

## 4.2 Vnitřní bufer

Vnitřní bufer je v automatu implementován jako dvouúrovňový, metody GetData a SetData mohou přistupovat pouze k buferu vyšší úrovně. K buferu nižší úrovně mohou přistupovat pouze privátní metody objektu zařízení při bitovém čtení nebo zápisu metody BitFromBuffer a BitToBuffer. Hodnota buferu vyšší úrovně je do buferu nižší úrovně zapisována při přechodu automatu do stavu LOAD_DATA, tj. při přípravě dalšího datového bajtu k odeslání na zařízení Master. Naopak hodnota buferu nižší úrovně je do buferu vyšší úrovně zapisována ve stavu ACCEPT_DATA před přechodem do stavu CLEAR_DATA (v době, kdy je přijat kompletní datový bajt ze zařízení Master).

Na obrázku 15 je pro přehlednost znázorněn způsob výměny dat mezi buferem nižší úrovně a bufery vyšší úrovně.




Obrázek 15: Implementace buferů třídy I2C

## 4.3 Popis atributů a metod

Tato podkapitola stručně popisuje význam jednotlivých metod a atributů třídy I2C.

### 4.3.1 Veřejné atributy a metody

| | |
|---|---|
| I2C(); | inicializuje objekt zařízení Slave |
| I2C(UCHAR Address, UCHAR Num); | dtto a nastaví adresu Slave zařízení a pořadí |
| ~I2C(); | destruktor |
| UCHAR GetData; | vrací data uložena v buferu |
| **void** SetData(UCHAR Data); | zapíše data do buferu |

| | |
|---|---|
| UCHAR SCL(); | vrací stav linky SCL (LOW, HIGH) |
| **void** SCL(UCHAR Level); | nastavuje linku SCL (LOW, HIGH) |
| UCHAR SDA(); | vrací stav linky SDA (LOW, HIGH) |
| **void** SDA(UCHAR Level); | nastavuje linku SDA (LOW, HIGH) |
| **void** SetSlaveAdrress(UCHAR Address); | nastavuje adresu Slave zařízení |
| UCHAR GetSlaveAdrress(); | vrací adresu Slave zařízení |
| **void** Reset(); | resetuje zařízení Slave a přejde do IDLE stavu |
| **bool** IsDataReady(); | vrací stav příznaku platnosti přijatých dat |
| **bool** IsNewState(); | vrací informaci, zda došlo ke změně stavu |
| **char** ∗ GetDeviceName(); | vrací jméno zařízení |
| **void** SetDeviceName(**char** ∗ Name); | nastavuje jméno zařízení |
| UCHAR GetDeviceNum(); | vrací pořadové číslo zařízení |
| SlaveState GetState(); | vrací aktuální stav automatu |

### 4.3.2 Privátní atributy a metody

| | |
|---|---|
| **void** BitToBuffer() ; | zapíše bit do buferu na pozici mBitCount |
| UCHAR BitFromBuffer(); | vrátí bit z buferu na pozici mBitCount |
| UCHAR mSCL; | stav linky SCL |
| UCHAR mSDAin; | stav linky SDA pro vstup |
| UCHAR mSDAout; | stav linky SDA pro výstup |
| UCHAR mOwnAddress; | adresa Slave zařízení |
| UCHAR mBuffer; | low level bufer (in/out) |
| UCHAR mDataIn; | high level bufer pro přijatá data |
| UCHAR mDataOut; | high level bufer pro data k odeslání |
| UCHAR mBitCount; | čítač pro in/out bufer |
| **bool** mDataReady; | indikuje platný přijatý datový bajt |
| **char** mSlaveName[20]; | název Slave zařízení (např. PCF8574) |
| UCHAR mDeviceNum; | obsahuje pořadové číslo zařízení |
| **bool** mNewState; | indikuje změnu stavu automatu |
| SlaveState EvalState(); | vyhodnotí a vrátí nový stav zařízení |
| SlaveState mState; | stav automatu |

## 4.4 Použití třídy I2C

Nyní si ukážeme způsob vytvoření dvou objektů PCF8574 a PCF9634 jako simulovaných Slave zařízení. Zařízením jsou přiděleny adresy `0x44` a `0x74` a pořadová čísla `0` a `1`. Objekty zařízení se jednoduše vytvoří zavoláním konstruktorů s předáním příslušných parametrů.

```
1 int main() {
2   I2C PCF8574(0x44, 0);
3   PCF8574.SetDeviceName("PCF8574");
4
```

```
5   I2C PCF9634(0x74, 1);
6   PCF9634.SetDeviceName("PCF9634");
7 }
```

Výpis 1: Vytvoření objektů simulovaných zařízení

Po vytvoření objektu zařízení Slave je tento objekt připraven ke své funkci. Zařízení je po vytvoření ve stavu IDLE a čeká na vnější signály (volání metod SCL, SDA).

```
1 // Nastavi na lince SCL uroven logicke jednicky
2 PCF8574.SCL(1);
3
4 // Nastavi na lince SDA uroven logicke nuly
5 PCF8574.SDA(0);
6
7 // Precte stav linky SDA
8 UCHAR level = PCF8574.SDA();
```

Výpis 2: Přístup k linkám SCL a SDA

Výhodou objektového řešení je možnost sdružení objektů reprezentujících různá Slave zařízení do pole, např. vektoru třídy std :: vector a možnosti jednoduchého volání metody SCL všech objektů vložených do pole. Tato implementace přibližuje přístup k jednotlivým zařízením obdobným způsobem jaký je použit u skutečného zapojení obvodů, kde jsou obvody svými linkami SCL a SDA elektricky propojeny paralelně.

```
1 // Vektor pro obsluhovana I2C zarizeni
2 std :: vector<I2C> mDevices;
3
4 // Iterator k prochazeni prvku vektoru
5 std :: vector<I2C>::iterator dev;
6
7 // Vlozime objekty obou zarizeni do vektoru
8 mDevices.push_back(PCF8574);
9 mDevices.push_back(PCF9634);
10
11 // Nastavime u vsech zarizeni na sbernici linku SCL na logickou jednicku
12 for (dev = mDevices.begin(); dev != mDevices.end(); dev++) {
13   dev->SCL(1);
14 }
```

Výpis 3: Ukázka volání metody SCL všech zařízení na sběrnici

Metoda GetState vrací aktuální stav zařízení dle popisu uvedeného na straně 21 v podkapitole 3.4. Třída I2C definuje k tomuto účelu výčtový typ SlaveState popisující jednotlivé stavy zařízení.

```
1 typedef enum SlaveState {
2   IDLE,
3   START,
```

```
4   ACCEPT_ADDRESS,
5   CLEAR_DATA,
6   LOAD_DATA,
7   SEND_DATA,
8   ACCEPT_DATA
9 };
```

Výpis 4: Výčtový typ pro stavy zařízení

V aplikaci je tedy možno kdykoli zjistit aktuální stav zařízení voláním metody GetState na příslušném objektu.

```
1 if (PCF8574.GetState() == LOAD_DATA) {
2   // Zarizeni pripravuje data pro odeslani
3 }
```

Výpis 5: Příklad použití metody GetState

Dalšími důležitými metodami třídy jsou metody GetData a SetData, pomocí kterých může aplikace přistupovat k datům uloženým ve vnitřním buferu zařízení. Vnitřní bufer využívá automat ve stavech CLEAR_DATA a ACCEPT_DATA v případě příjmu dat od zařízení Master a ve stavech LOAD_DATA a SEND_DATA v případě zasílání dat zařízení Master.

K přečtení obsahu buferu stačí tedy zavolat metodu GetData. Třída I2C ještě umožňuje prostřednictvím metody IsDataReady zjistit, zda jsou v buferu nově přijatá data. Metoda IsDataReady testuje stav vnitřního atributu mDataReady a vrací jeho hodnotu. Po volání této metody je příznak mDataReady automaticky resetován.

```
1 if (PCF8574.IsDataReady()) {
2   // Jsou nova data, provedeme cteni z buferu
3   UCHAR ReceivedData = PCF8574.GetData();
4 }
```

Výpis 6: Ukázka čtení dat ze zařízení

Zápis dat na zařízení Slave je uskutečňován, jak bylo zmíněno výše, prostřednictvím metody SetData(data). Jak bylo diskutováno v podkapitole 4.2 na straně 23, data je možné do zařízení zapsat kdykoli, nehledě na aktuální stav zařízení.

```
1 // Zapis dat do buferu
2 PCF8574.SetData(0x3C);
```

Výpis 7: Ukázka zápisu dat do zařízení

# 5 Popis obvodů vybraných pro funkci simulátoru

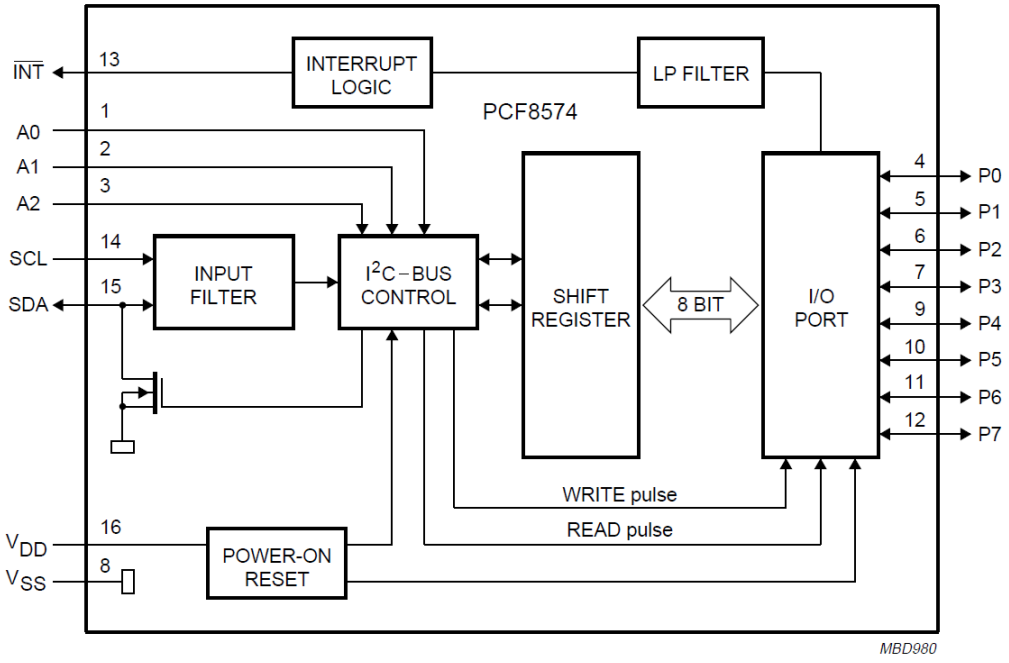
Dle specifikace zadání bakalářské práce má být vytvořen programový simulátor obvodu PCF8574 a AD převodníku. Typ obvodu analogově-digitálního převodníku nebyl specifikován a proto bude pro účely simulace navržen obecný 8bitový jednokanálový AD převodník s možností změny adresy a nastavením referenčního napětí.

## 5.1 PCF8574 I/O expandér pro sběrnici I²C

PCF8574 je integrovaný obvod, který obsahuje kvazi–obousměrný port o šířce 8 bitů a rozhraní I²C. Tento obvod je využíván v elektronických zařízeních nejčastěji pro rozšíření vstupně-výstupních linek mikrořadičů nebo jiných obvodů. Obvod má nízkou spotřebu energie (vyroben technologií CMOS) a jeho výstupy mohou řídit LED diody (s omezovacími rezistory). Obvod obsahuje také výstup $\overline{\mathbf{INT}}$ (INTerrupt), který může být použit jako zdroj přerušení pro jiné obvody, nejčastěji mikrořadič. Pomocí tohoto výstupu může být mikrořadič informován o změně dat na portu (na linkách využívaných jako vstupy) bez nutnosti opakovaně číst obsah celého portu. Každá z osmi I/O linek může být nezávisle používána jako vstupní nebo výstupní.

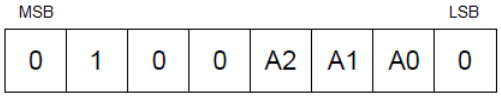
Vzhledem k tomu, že pro účely simulace obvodu PCF8574 nemá výstup $\overline{\mathbf{INT}}$ význam, bude jeho funkce v řešeném simulátoru vynechána.

Na obrázku 16 je zobrazeno blokové schéma obvodu PCF8574. Podrobný popis obvodu PCF8574 lze nalézt v datovém listu [2].




Obrázek 16: Blokové schéma obvodu PCF8574 (převzato z [2])

Pro identifikaci obvodu na sběrnici $I^2C$ má obvod výrobcem přidělenou adresu `0x40` (dekadicky 64). Tuto adresu je možné prostřednictvím třech adresních vstupů `A0..A2` upravit podle potřeby. Adresní vstupy umožňují, aby na jedné sběrnici mohlo pracovat nezávisle až osm těchto obvodů. Na obrázku 17 je uveden formát adresy.




Obrázek 17: Formát adresy obvodu PCF8574

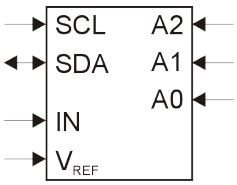
Zápis dat na port je v obvodu při přenosu datového paketu prováděn s náběžnou hranou devátého cyklu hodin na lince SCL (viz časový diagram v příloze B), kdy dojde k přepisu hodnoty uložené ve vnitřním registru obvodu na výstupní port. Čtení stavu vstupních linek a zápisu hodnot do vnitřního registru je prováděno taktéž s náběžnou hranou devátého cyklu hodin na lince SCL (viz časový diagram v příloze B). Na linky portu, které budou používány jako vstupní, musí být před jejich použitím (čtením) zapsána logická jednička.

## 5.2 AD převodník

Pro účely simulace analogově-digitálního převodníku budeme vycházet z imaginárního obvodu, pro který si stanovíme následující vlastnosti:

- rozlišení 8 bitů,
- jeden vstupní kanál,
- nastavitelný rozsah referenčního napětí ($V_{REF}$),
- vstupní napěťový rozsah 0V ... $V_{REF}$,
- nastavitelná adresa zařízení v rozsahu dolních třech bitů (`A0..A2`).

Blokové schéma imaginárního AD převodníku odpovídající uvedeným vlastnostem je na obrázku.




Obrázek 18: Blokové schéma AD převodníku

Simulovaný AD převodník bude konvertovat vstupní napěťovou (analogovou) úroveň definovanou uživatelem na číselné vyjádření s rozlišením 8 bitů. Oproti reálnému AD

převodníku nebude mít definovanou dobu převodu. Převod bude prováděn ihned po změně napěťové úrovně ze strany uživatele bez závislosti na tom, zda byl obvod zařízením Master osloven či nikoli. Zařízení Master, které bude adresovat AD převodník s následným čtením dat, obdrží aktuální hodnotu převodu uloženou v buferu zařízení.

Převod napěťové úrovně na číselné vyjádření

$$ADC = \frac{V_{IN} \cdot 255}{V_{REF}} \tag{1}$$

kde $V_{IN}$ je vstupní napětí a $V_{REF}$ je nastavené referenční napětí. Analogovou zem (0 voltů) reprezentuje hodnota `0x00` a referenční napětí hodnota `0xFF`.

Pro identifikaci na sběrnici I$^2$C přidělíme obvodu výchozí adresu na `0x60` (dekadicky 96).

# 6 Návrh a popis grafického rozhraní simulátoru

Úkolem grafického simulátoru je uživateli poskytnout v grafické podobě možnosti, které nabízejí reálné obvody, jejíž funkci simulujeme. S ohledem na to, že aplikace bude hlavně využívána při vývoji a ladění klientských aplikací, bude praktické, aby aplikace simulátoru rovněž uživateli poskytovala informace o stavu logických úrovní na sběrnici $I^2C$ samotné. Tyto informace budou pro uživatele neocenitelné v průběhu ladění v režimu Step–By–Step. Aplikace by rovněž měla uživateli umožnit v průběhu chodu úpravu adresy simulovaných zařízení.

Mezi základní funkce, které aplikace má nabízet, bude u obvodu PCF8574 tedy patřit zobrazení přijatého datového bajtu v binární podobě vyjádřenou graficky pomocí LED diod a nastavení logické hodnoty na jednotlivých vstupních pinech portu. U obvodu AD převodníku to bude především nastavení vstupního a referenčního napětí.

K vývoji grafického rozhraní aplikace bylo využito multiplatformní knihovny GTKmm, což je objektová varianta knihovny GTK+ pro jazyk C++. Knihovna nabízí podporu pro platformy GNU[3]/Linux, Unix, Windows a Mac OS X. Vývoj aplikace je při použití možností této knihovny snadnější a rychlejší. Knihovna nabízí řadu ovládacích prvků pro grafické prostředí, tzv. widgetů, podporu pro práci s vlákny, grafikou, řetězci v kódování UTF8 atd. GTK+ je volně šiřitelný software, který je součástí projektu GNU. Licenční podmínky pro GTK+ umožňují použití knihovny při vývoji softwaru bez jakýchkoliv licenčních nebo jiných poplatků.

## 6.1 Vzhled aplikace

Grafické rozhraní tvoří jedno okno, které bude zobrazovat všechny dostupné informace, tj. informace o stavu linek sběrnice, adresních bitech, hodnoty vstupu a výstupu simulovaných zařízení a informace o připojeném klientovi. Rozsah informací viditelných v okně bude uživatelsky nastavitelný, uživatel si bude moci zvolit, která zařízení a které informace budou v okně viditelné. Přístup k volbám aplikace bude proveden prostřednictvím nabídky v horní části okna. Vzhled okna aplikace při maximálním zobrazení je na obrázku 19. Na obrázku 20 je vzhled okna při minimálním zobrazení, jak pro zařízení PCF8574, tak AD převodníku.

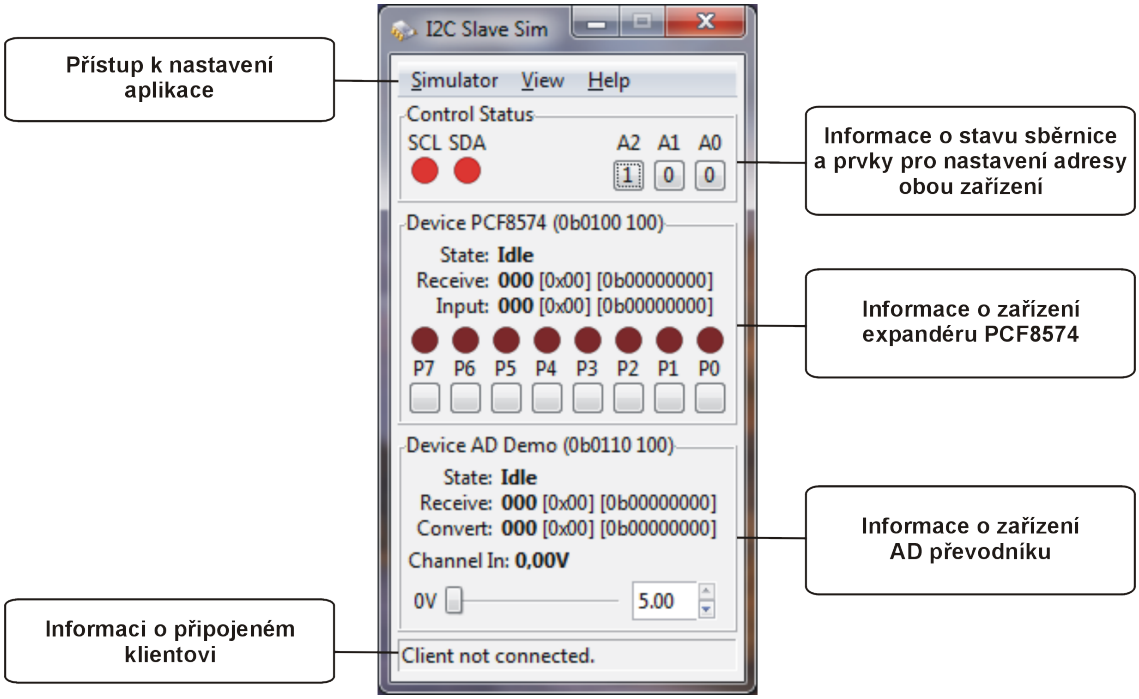
U obou zařízení bude možno nezávisle uživatelsky nastavovat zobrazení hodnoty na vstupu a výstup a také doplňkovou informaci o vnitřním stavu zařízení (automatu).

Barvu LED diody (barvy pro diodu rozsvícenou i zhasnutou) lze nastavit v konfiguračním souboru aplikace. Do konfiguračního souboru jsou ukládány rovněž nastavení provedená v GUI aplikace, např. viditelnost jednotlivých rámů okna, stavového řádku atd. Detailní popis konfiguračních nastavení je v kapitole 7.5.

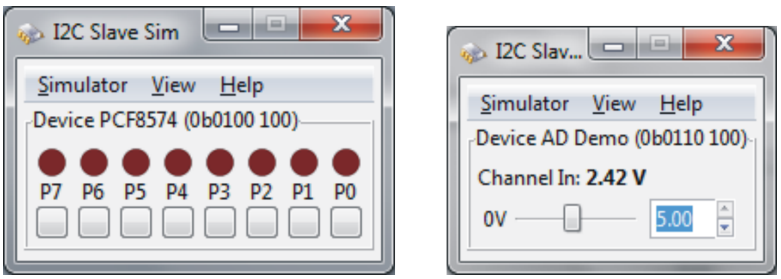
### 6.1.1 Nabídka aplikace

Nabídka aplikace obsahuje jednak volby pro úpravu zobrazení simulátoru (menu `View`) a volby vztahující se k aplikaci jako takové (menu `Simulator`). V menu `Simulator`,

[3]GNU je projekt zaměřený na svobodný software, inspirovaný operačními systémy unixového typu.

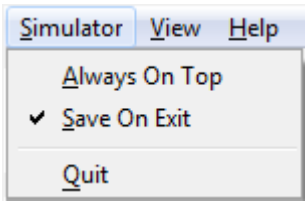



Obrázek 19: Vzhled hlavního okna aplikace




Obrázek 20: Ukázka vzhledu oken v minimalizovaném režimu

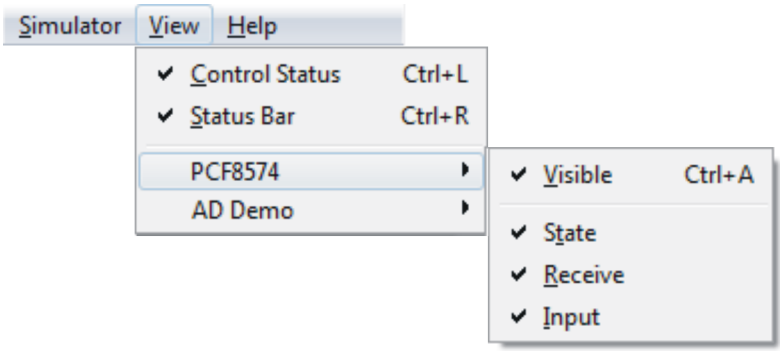
viz obrázek 21, jsou dostupné volby umožňující ponechat okno aplikace vždy nahoře, ukládat nastavení aplikace při ukončení a volba pro ukončení aplikace.




Obrázek 21: Položky menu Simulator

Prostřednictvím menu `View` je možné nastavovat viditelnost rámů v GUI rozhraní a jejich informační obsah. Aplikace nedovolí skrytí rámů obou zařízení současně, musí být tedy vždy vybráno alespoň jedno z nich. Dostupné volby jsou zobrazeny na obrázku 22.

Menu `Help` obsahuje pouze možnost vyvolání dialogového okna s informacemi o aplikaci.




Obrázek 22: Položky menu View

### 6.1.2 Ovládací prvky stavu sběrnice a adresy

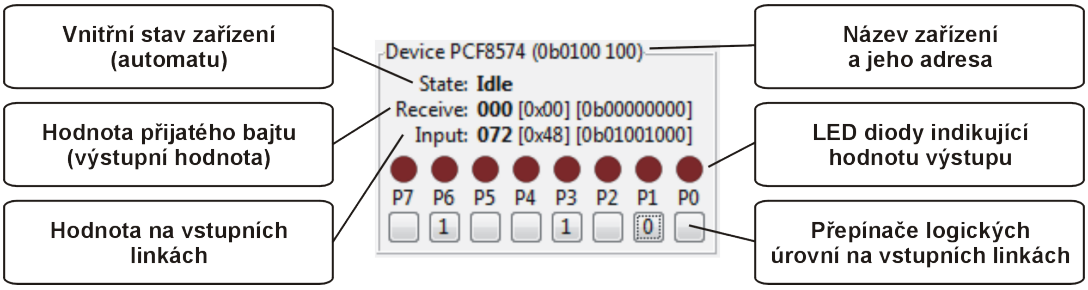
V okně aplikace je stav sběrnice ve zvláštním rámu s titulkem „Control Status“, kde je zobrazen logický stav linek SCL a SDA pomocí imitace LED diod. Ve stejném rámu jsou také umístěna dvoustavová tlačítka, pomocí kterých je možné nastavit dolní tři bity (`A0..A2`) adresy u obou simulovaných zařízení současně. Tlačítko na svém titulku zobrazuje nastavenou logickou hodnotu symbolem 1 nebo 0. Změna adresy je u obou zařízení provedena ihned, nové adresy zařízení jsou však zohledněny až po vyslání nové podmínky START (nebo REPEATED START) ze strany zařízení Master. Pokud byly tedy adresy změněny v době aktivního přenosu, tj. mezi podmínkami START a STOP, nemá tato změna na probíhající přenos vliv.

Indikační LED diody pro signály SCL a SDA budou spíše využitelné při činnosti klientské aplikace v režimu ladění, neboť v dynamickém režimu nebude jejich indikace vzhledem k rychlosti změny stavu linek viditelná.

### 6.1.3 Ovládací prvky zařízení PCF8574

Ovládací a zobrazovací prvky pro obvod PCF8574 jsou v okně aplikace umístěny v rámu označeném názvem obvodu, za kterým následuje údaj o jeho nastavené adrese v binárním vyjádření. Po změně adres zařízení provedené v rámu „Control Status“, je tato změna v titulku rámu ihned zohledněna.

Plně rozvinutý rám obsahuje nejprve informační oblast s údaji o vnitřním stavu zařízení, výstupní hodnotě na portu zařízení a hodnotě na vstupních linkách. Viditelnost jednotlivých informačních řádků lze upravovat prostřednictvím nabídky aplikace. Hodnota na výstupním portu zařízení je také graficky indikována pomocí imitace LED diod

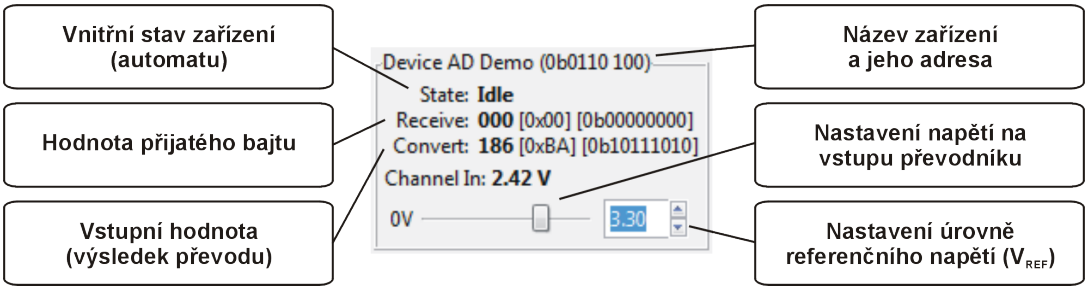



Obrázek 23: Popis prvků rámu zařízení PCF8574

uspořádaných zleva od bitu s nejvyšší váhou (P7) po bit s nejnižší váhou (P0). Pod pásem LED diod jsou umístěna třístavová tlačítka umožňující nastavení hodnoty na jednotlivých linkách v případě, že budou používány jako vstupní. Opakovaným stiskem tlačítka lze přecházet mezi stavy logická úroveň $1$ $\Rightarrow$ logická úroveň $0$ $\Rightarrow$ vysoká impedance (Hi-Z). Najetím ukazatele myši nad tlačítko je zobrazena dekadická hodnota příslušné linky (formou tooltipu).

### 6.1.4 Ovládací prvky AD převodníku

Druhým simulovaným obvodem je AD převodník, jehož rám ve svém titulku obsahuje (obdobně jako rám expandéru PCF8574) informaci o typu obvodu a jeho nastavené adrese. Vzhledem k tomu, že typ obvodu je pouze imaginární, jeho název byl nastaven na „AD Demo“. Plně rozvinutý rám obsahuje informační oblast s údaji o hodnotě přijatého datového bajtu, údaj o stavu vnitřního registru obsahujícího výsledek převodu a informaci o vnitřním stavu zařízení (automatu).



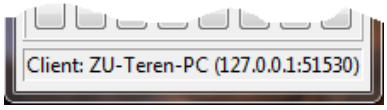

Obrázek 24: Popis prvků rámu AD převodníku

Ovládací prvky rámu dále tvoří horizontální posuvník, pomocí kterého lze upravovat hodnotu vstupního napětí a spin box pro úpravu hodnoty referenčního napětí ($V_{REF}$). Vstupní napětí přiváděné na vstup převodníku pomocí posuvníku je v rozsahu $0$ V $\ldots$ $V_{REF}$. V závislosti na hodnotě referenčního napětí je upravována hodnota vstupního napětí. Hodnotu referenčního napětí lze nastavovat v rozsahu $1$ až $5$ voltů. Výsledkem převodu vstupního napětí, dle vzorce (1), je číslo v rozsahu $0 \ldots 255$ (`0x00` $\ldots$ `0xFF`).

Po změně vstupního napětí je okamžitě proveden převod a jeho výsledek je zapsán do vnitřního buferu.

### 6.1.5 Stavový řádek

Stavový řádek aplikace slouží pouze k zobrazení informací o připojeném klientovi. Po připojení klienta je ve stavovém řádku zobrazen název počítače a IP adresa klienta včetně čísla portu, na kterém probíhá komunikace. Stavový řádek lze prostřednictvím nabídky aplikace skrýt.



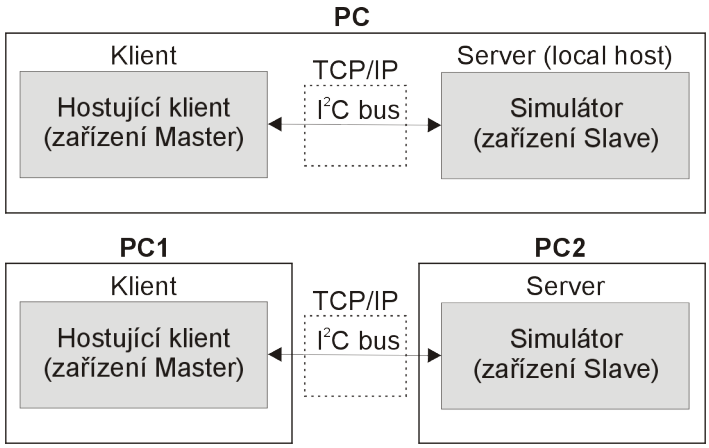

Obrázek 25: Stavový řádek aplikace

## 6.2 Komunikace simulátoru s klientem

Aplikace simulátoru, která plní roli dvou Slave zařízení připojených na sběrnici $I^2C$, musí být nějakým způsobem na této sběrnici „propojena“ se zařízením typu Master (klientskou aplikací). Při volbě způsobu spojení bylo rozhodnuto o použití soketů a transportního protokolu TCP. Volba tohoto typu spojení přináší několik výhod a zároveň snadnou implementaci komunikačního rozhraní a obousměrného přenosu zpráv mezi oběma aplikacemi. Volba protokolu TCP je výhodná pro tento typ aplikací z několika hledisek. Předně protokol zajišťuje spolehlivý přenos dat mezi aplikacemi, je tedy zajištěno, že všechna data budou vždy doručena. Navázáním spojení mezi aplikacemi je vytvořen komunikační kanál, který je plně duplexní, tj. umožňuje zasílání a příjem dat oběma stranami nezávisle. Implementace protokolu TCP v aplikaci je usnadněna použitím knihovny `WinSock`.

Obě aplikace pracují v režimu Klient–Server. Mezi výhody použití architektury Klient–Server lze zařadit možnosti spojení a provozu obou aplikací jak v rámci jednoho počítače, tak i na nezávislých počítačích zapojených do sítě. V danou chvíli je umožněno vytvoření platného spojení pouze s jedním klientem. V případě, že se pokouší o připojení k simulátoru další klient, je jeho požadavek na spojení odmítnut (klient obdrží příslušnou zprávu). V průběhu činnosti simulátoru tedy server nepřetržitě (tj. i v případě aktivní komunikace s již připojeným klientem) naslouchá příchozím žádostem o spojení.

Aplikace simulátoru pracuje v roli serveru, klientská (hostující) aplikace v režimu klienta. Simulátor poslouchá na zvoleném portu a čeká na hostujícího klienta. Po úspěšném navázání spojení jsou všechna simulovaná zařízení (Master a Slave) „propojena“ na jedné sběrnici $I^2C$ a mohou navzájem komunikovat. Vytvoření spojení mezi aplikacemi plně simuluje chování reálné sběrnice $I^2C$. Mezi aplikacemi jsou vyměňovány pouze zprávy týkající se nastavení nebo čtení stavu linek SCL a SDA. Varianty spojení ukazuje obrázek 26.




Obrázek 26: Varianty spojení simulátoru s klientem

## 6.3 Komunikační protokol

Pro účely komunikace mezi klientem a simulátorem byl vytvořen jednoduchý protokol. Pomocí tohoto protokolu budou mezi oběma aplikacemi posílány zprávy. Na konci každé zprávy je vždy znak ukončovací (hodnoty `0x00`). Klient bude simulátoru zasílat zprávy o změně stavu linky SCL nebo SDA anebo požadavek na čtení stavu linky SDA. Server bude klientovi zasílat zprávy o stavu linky SDA a zprávy o úspěšném připojení nebo odmítnutí klienta.

Zprávy klienta jsou vždy složeny z příkazové části následované názvem linky. Obě části zprávy jsou odděleny znakem `_` (podtržítko).

Klient bude tedy odesílat zprávy ve formátu `XXX_YYY` zakončené znakem `\n` (`0x00`), kde `XXX` je SET, RST nebo GET, `YYY` je SDA nebo SCL. Přehled všech kombinací zpráv užívaných klientem je v tabulce 5.

| Zpráva | Popis |
|---|---|
| SET_SCL | Nastaví linku SCL na úroveň logické jedničky |
| SET_SDA | Nastaví linku SDA na úroveň logické jedničky |
| RST_SCL | Nastaví linku SCL na úroveň logické nuly |
| RST_SDA | Nastaví linku SDA na úroveň logické nuly |
| GET_SCL | Požadavek na stav linky SCL |
| GET_SDA | Požadavek na stav linky SDA |

Tabulka 5: Druhy zpráv klienta

Ze strany serveru budou odesílány dva druhy zpráv. Jednak jsou to zprávy o úspěšnosti připojení klienta a pak zprávy o stavu linek Slave zařízení.

Zprávy o stavu linek zařízení Slave budou ve formátu `YYY=Z` zakončené znakem `\n`, kde `YYY` je SDA nebo SCL a `Z` je 0 nebo 1. Přehled všech kombinací zpráv užívaných serverem je v tabulce 6.

| Zpráva | Popis |
|---|---|
| SDA=1 | Stav linky SDA je v úrovni logické jedničky |
| SDA=0 | Stav linky SDA je v úrovni logické nuly |
| SCL=1 | Stav linky SCL je v úrovni logické jedničky |
| SCL=0 | Stav linky SCL je v úrovni logické nuly |

Tabulka 6: Druhy zpráv serveru

Druhým typem zpráv, které bude server odesílat jsou zprávy o úspěšnosti připojení klienta k simulátoru (viz tabulka 7).

| Zpráva | Popis |
|---|---|
| CONNECTED! OK... | Spojení ustaveno, je možné zahájit komunikaci |
| DECLINED. SORRY. | Spojení odmítnuto, je již připojen jiný klient |

Tabulka 7: Druhy zpráv serveru (spojení)

## 6.4 Aplikace klienta

K tvorbě klientských aplikací v prostředí jazyka C, příp. C++, byla vytvořena knihovna `I2C_Client`, která umožňuje jednoduchou komunikaci se simulátorem. Všechny funkce knihovny jsou pojmenovány s prefixem i2c_.

Knihovna obsahuje nízkoúrovňové funkce pro práci s linkami SCL a SDA a funkce pro inicializaci a ukončení komunikace se simulátorem. Knihovna rovněž nabízí funkce vyšší úrovně pro práci na sběrnici $I^2C$, které jsou implementovány tak, aby programátorovi usnadnily použití sběrnice. Tyto funkce pouze sdružují funkce nižší úrovně za účelem provedení určité činnosti na sběrnici, např. vyslání podmínky START. Seznam funkcí vyšší úrovně je v tabulce 8.

Knihovna také definuje datový typ i2cLevel, který je možné používat při deklaraci proměnných vystupujících jako logické hodnoty. Jakákoliv hodnota, vyjma 0, je funkcemi považována za logickou jedničku. V klientské aplikaci je možné také použít předdefinovaných konstant FLAG_R = 1 a FLAG_W = 0 při adresaci zařízení.

Pro přístup k linkám SCL a SDA knihovna obsahuje speciální stejnojmenné proměnné (třídního typu) s přetíženým operátorem „=“.

K využití funkcí knihovny postačuje v klientské aplikaci importovat (zahrnout) knihovnu `I2C_Client`. Knihovna obsahuje funkci i2c_SimInit() nebo přetíženou variantu i2c_SimInit(**char** * Host, **int** Port), která zajistí spojení se simulátorem. Pokud se spojení nezdaří, vrací funkce hodnotu `false`, v opačném případě `true`. Pro legitimní ukončení spojení se simulátorem má knihovna k dispozici funkci i2c_SimClose().

Při použití funkce i2c_SimInit() je k vytvoření spojení se simulátorem použito implicitních hodnot konstant DEFAULT_HOST_NAME = "localhost" a DEFAULT_PORT = 3000, které jsou deklarovány v hlavičkovém souboru knihovny. Tyto konstanty je možné dle potřeby upravit. Přetížená varianta funkce umožňuje volat funkci s vlastními hodnotami. Obě

funkce v případě chybových stavů vypisují příslušnou informaci v konzolovém okně aplikace.

Příklad využití knihovny v klientské aplikaci je uveden ve výpisu kódu 8.

```
1  #include "I2C_Client.h"
2
3  int main() {
4    // Inicializace spojeni (mistni PC, port 3000)
5    if (! i2c_SimInit ("localhost", 3000)) return 1;
6
7    // Ukazka prace s funkcemi pro primy pristup k linkam SCL a SDA
8    // Zapis na linky
9    SCL = 1;
10   SDA = 1;
11   SCL = 0;
12   SCL = 1;
13
14   // Cteni stavu linek
15   i2cLevel x = SCL;
16   i2cLevel y = SDA;
17
18   while (true) {
19     // Nyni pouziti funkci vyssi urovne
20     // Vyslani podminky START
21     i2c_SendSTART();
22
23     // Zapis adresy zarizeni Slave na sbernici
24     i2c_SendData(SlaveAddress + FLAG_W);
25
26     // Zjistime stav ACK
27     if (i2c_GetAck() == 1) {
28       // Adresa neprijata -> STOP
29       break;
30     }
31
32     for (int i = 0; i < 8; i++) {
33       i2c_SendData(1 << i);
34       printf ("M: Odeslana data: %d\n", 1 << i);
35
36       // Zjistime, co na odeslana data rika Slave
37       if (i2c_GetAck() == 1) {
38         // Data neprijata nebo slave ukoncuje prijem -> STOP
39         break;
40       }
41       delay_ms(250); // nebo Sleep(250);
42     }
43   } // end while
44
45   // Vyslani podminky STOP
46   i2c_SendSTOP();
47
48   // Ukonceni spojeni se simulatorem
```

```
49    i2c_SimClose();
50 }
```

Výpis 8: Příklad použití knihovny (posun LED od LSB k MSB)

| **Název funkce** | **Popis** |
|---|---|
| i2c_SendSTART() | Master vyšle na sběrnici podmínku START |
| i2c_SendSTOP() | Master vyšle na sběrnici podmínku STOP |
| i2c_SendData(UCHAR Data) | Master vyšle na sběrnici Data |
| i2c_SendAck() | Master potvrdí přijetí dat, tj. nastaví ACK |
| i2c_SendNack() | Master nepotvrdí přijetí dat, resp. sděluje, že další data nepotřebuje (nastaví NACK) |
| UCHAR i2c_ReceiveData() | Přečte a vrátí ze zařízení Slave datový bajt |
| UCHAR i2c_GetAck() | Zjistí stav linky SDA na straně Slave zařízení a vrací 0 pokud je ACK a 1 pokud je NACK |

Tabulka 8: Funkce vyšší úrovně knihovny I2C_Client

# 7 Realizace grafického rozhraní

Aplikace simulátoru byla vytvořena plně objektovým způsobem. Po startu aplikace je ve funkci main() vytvořen objekt mainWin třídy MainWindow, která je potomkem třídy Gtk ::Window. Objekt mainWin při svém vzniku provede kompletní inicializaci aplikace. Po dokončení inicializace je volána metoda run, které je předán ukazatel na objekt mainWin, čímž dojde ke spuštění hlavní smyčky okna. Poté již aplikace vyčkává na vnější události či události způsobené uživatelem prostřednictvím GUI.

Rozbor dílčích částí aplikace a jednotlivých vláken je uveden v následujících podkapitolách.

```
1  int main(int argc, char * argv[]) {
2
3    // Inicializace systemu vlaken
4    Glib :: thread_init () ;
5
6    Gtk::Main MainGUI(argc, argv);
7
8    // Vytvorime objekt hlavniho okna
9    MainWindow mainWin;
10
11   // Volanim metody Run je spustena smycka hlavniho okna a bezi
12   // do doby nez je okno zavreno
13   Gtk::Main::run(mainWin);
14
15   return 0;
16 }
```

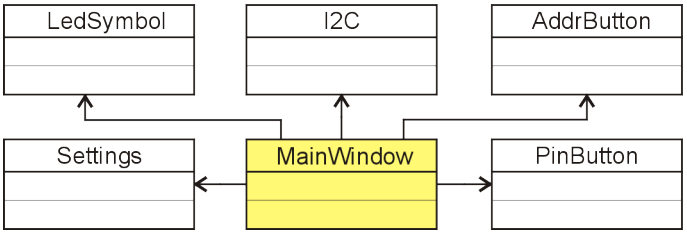
Výpis 9: Inicializace aplikace

## 7.1 Objektová koncepce

Jádro aplikace reprezentuje objekt třídy MainWindow, který obsahuje samotné GUI aplikace, obsluhuje přicházející události, zajišťuje překreslování zobrazovacích prvků a výměnu zpráv s připojeným klientem atd. Objekt MainWindow při své činnosti spolupracuje s dalšími specializovanými objekty založenými na třídách

- I2C (zařízení Slave),
- LedSymbol (grafická podoba LED diody),
- AddrButton (tlačítko pro vstupní pin adresy),
- PinButton (tlačítko pro vstupní pin portu),
- Settings (konfigurace aplikace).

Na obrázku 27 je uveden zjednodušený třídní diagram aplikace simulátoru. Úplný třídní diagram s uvedením atributů a metod je uveden v příloze C.




Obrázek 27: Zjednodušený třídní diagram aplikace

## 7.2 Inicializace aplikace

Tak jak bylo zmíněno v kapitole 7, aplikace po svém spuštění vytváří objekt mainWin založený na třídě MainWindow. V konstruktoru této třídy je provedena řada činností počínaje inicializací všech objektů GUI rozhraní (widgetů) a nabídky aplikace, načtení konfiguračních údajů z inicializačního souboru (pomocí objektu třídy Settings), vytvoření vektoru s objekty simulovaných $I^2C$ zařízení, napojení obslužných metod na události. Poslední důležitou operací prováděnou při inicializaci objektu je vytvoření nového vlákna ServiceThread, které je reprezentováno metodou MainWindow::ServiceThread().

Vlákno ServiceThread zahájí činnost ihned po svém vytvoření a zajišťuje několik důležitých činností – inicializaci rozhraní `WinSock`, vytvoření soketu pro naslouchání na zvoleném portu a obsluhu požadavků na připojení klientů. Detailní popis tohoto vlákna je uveden v části 7.3.2.

V konstruktoru třídy MainWindow je také provedeno připojení obslužných metod pro asynchronní události vznikající v jiných vláknech aplikace.

## 7.3 Vlákna

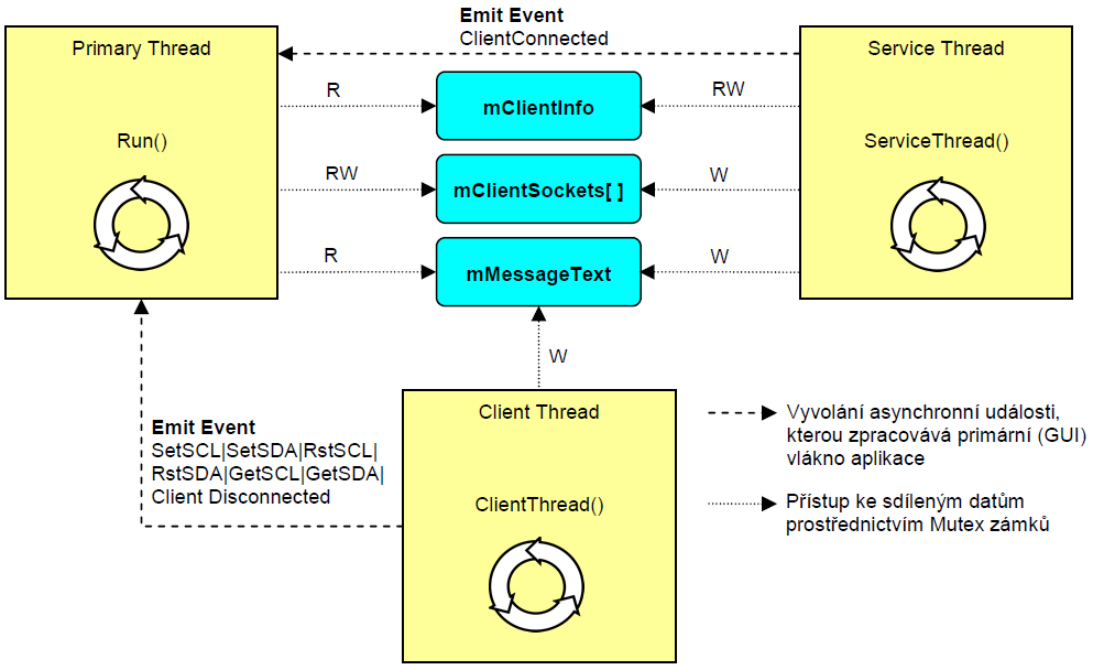
Aplikace během svého provozu vytváří vedle primárního vlákna další dvě pomocná vlákna. Jedním z nich je již zmíněné servisní vlákno ServiceThread, které je vytvořeno bezprostředně po spuštění aplikace.

Dalším vláknem je vlákno pro obsluhu připojeného klienta. Toto vlákno reprezentované metodou MainWindow::ClientThread() objektu mainWin zajišťuje výměnu zpráv mezi simulátorem a klientem. Jeho úkolem je interpretace přijatých zpráv a zasílání příslušných událostí primárnímu vláknu, které provádí obsluhu těchto událostí.

Vlákno ClientThread je vytvořeno primárním vláknem na základě žádosti vlákna ServiceThread po připojení prvního klienta k simulátoru a zůstává v činnosti až do okamžiku jeho odpojení, kdy je ukončeno.

Vlákna si mezi sebou vyměňují data pomocí sdílených atributů mClientInfo, mClient Sockets[] a mMessageText třídy MainWindow. Přístup vláken k těmto sdíleným atributům je prováděn s použitím mutex[4] zámků. Na obrázku 28 je zobrazena spolupráce všech tří vláken.

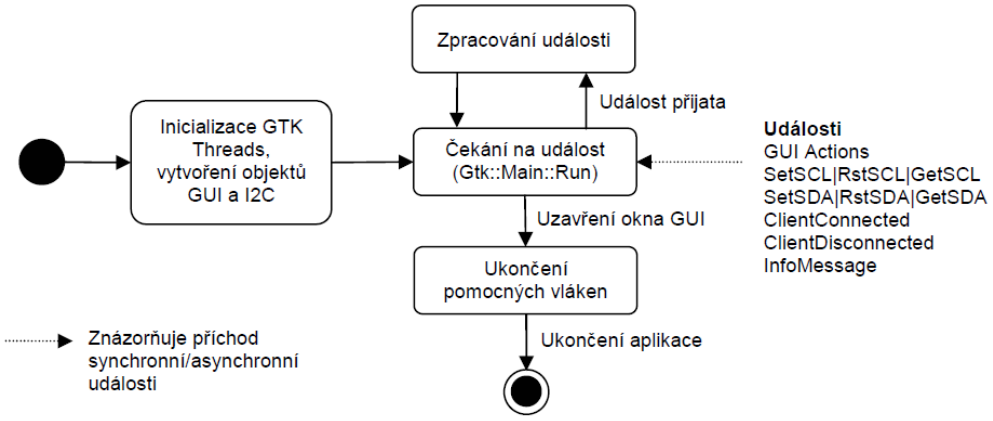
[4]MUTual EXclusion (vzájemné vyloučení) je algoritmus používaný v programování jako synchronizační prostředek v prostředí s více vlákny.




Obrázek 28: Komunikace mezi vlákny aplikace

### 7.3.1 Primární vlákno

V primárním vláknu je po inicializačních procedurách spuštěna hlavní smyčka zpráv okna, která zpracovává všechny události, tj. události vyvolané uživatelem prostřednictvím GUI rozhraní a také události vnitřní, vyvolané pomocnými vlákny. Primární vlákno je ukončeno v okamžiku uzavření aplikace, tj. poté kdy ukončí svou činnost hlavní smyčka zpráv. Na obrázku 29 je zobrazen stavový diagram primárního vlákna.




Obrázek 29: Stavový diagram primárního vlákna

Vzhledem k tomu, že události pomocných vláken (ClientThread a ServiceThread) mají vliv na zobrazení ovládacích prvků (widgetů) v GUI rozhraní aplikace, bylo nutno v aplikaci implementovat takový mechanismus zpracování zpráv, aby byl obslužný kód takové události vždy vykonán v rámci kódu primárního vlákna. Ovládací prvky GUI rozhraní (jejich atributy a metody) nejsou přizpůsobeny současnému provádění kódu prostřednictvím více vláken. Pokud by např. k atributu ovládacího prvku přistupovalo více vláken současně, vedlo by to k nepředvídatelným výsledkům, příp. až ke zhroucení aplikace.

V aplikaci byla pro komunikaci mezi vlákny použita třída Glib::Dispatcher. Pomocí této třídy je možné snadno implementovat mechanismus předávání zpráv z jednoho vlákna do jiného (v systémech Win32 se na pozadí používají mutex zámky). Na rozdíl od běžného zpracování událostí jsou oznámení mezi vlákny předávány asynchronně. Jedná se o jednoduchý a účinný způsob komunikace mezi vlákny a je zvláště výhodný pro aplikace s jediným vláknem pro GUI.

Způsob implementace je následující:

1. přijímající (primární) vlákno vytvoří objekt třídy Glib::Dispatcher; objekt musí být vytvořen dříve než vysílající vlákno (ClientThread a ServiceThread),

2. přijímající vlákno připojí na vytvořený objekt obslužnou metodu, která bude provedena po přijetí oznámení od jiného vlákna. Kód této metody bude *vždy* proveden v rámci primárního vlákna.

Výpis programu 10 ukazuje způsob implementace, pomocí kterého je zpracován příjem zprávy klienta požadujícího nastavení linky SCL na úroveň logické jedničky. Zpráva je přijata pomocným vláknem ClientThread, které tuto událost signalizuje pomocí metody emit() primárnímu vláknu a to provede příslušné změny v GUI rozhraní. Shodným způsobem je v aplikaci řešena obsluha událostí při příjmu ostatních zpráv klienta.

```
1  class MainWindow : public Gtk::Window {
2  public:
3    // ...
4
5  protected:
6    // ...
7
8  private:
9    Glib::Dispatcher mSetSCL;     // Objekt dispatcheru
10   Glib::Thread * mClientThread; // Ukazatel na vlakno
11
12   void on_set_scl();       // Obsluzna metoda udalosti
13   void ClientThread();     // Funkce klientskeho vlakna
14 }
15
16 MainWindow::MainWindow() {
17   // Pripojeni  obsluzne metody na objekt dispatcheru
```

```
18    mSetSCL
19      .connect(sigc::mem_fun(* this, &MainWindow::on_set_scl));
20
21    // Vytvoreni pomocneho vlakna ClientThread
22    mClientThread =
23      Glib :: Thread::create(sigc :: mem_fun(* this, &MainWindow::ClientThread), true);
24  }
25
26  void MainWindow::on_set_scl() {
27    // Kod teto metody provede pozadovane operace a muze bez
28    // problemu pristupovat k atributum a metodam ovladacich
29    // prvku v okne aplikace, nebot je vzdy provaden synchronne
30    // v primarnim vlaknu
31  }
32
33  // Metoda pomocneho vlakna ClientThread
34  MainWindow::ClientThread() {
35    while (true) {
36      // Cekani na zpravu
37      // ...
38      if (Message == "SET_SCL")) {
39          // Zasleme oznameni primarnimu vlaknu
40          mSetSCL.emit();
41      }
42    }
43  }
```

Výpis 10: Způsob komunikace mezi vlákny

Primární vlákno je tedy odpovědné za zpracování všech událostí, které vznikly jako důsledek přijetí zprávy klienta. Primární vlákno tedy zpracovává jednotlivé zprávy prostřednictvím samostatných objektů typu Glib::Dispatcher. Nyní si ukážeme zpracování dvou typů zpráv – zprávy RST_SCL a GET_SDA.

Výpis programu 11 ukazuje zpracování přijaté zprávy, kdy klient nastavuje linku SCL na úroveň logické nuly. Obslužná metoda nastaví úroveň linky SCL u všech simulovaných zařízení a zároveň zajistí překreslení LED diody signalizující stav linky. Metoda zároveň ověří, zda byl přijat kompletní datový bajt, a pokud ano, zajistí změnu úrovní LED diod indikující jeho hodnotu.

```
1  void MainWindow::on_reset_scl() {
2    vector<I2C>::iterator dev;
3
4    // Projdeme vsechna zarizeni ve vektoru a nastavime SCL=0
5    for (dev = mDevices.begin(); dev != mDevices.end(); dev++) {
6      dev->SCL(0);
7
8      // Zjistime , zda doslo ke zmene stavu zarizeni a pokud ano
9      // upravime text ve formulari
10     if (dev->IsNewState()) {
11       SetDeviceState(dev->GetDeviceNum(), dev->GetState());
12     }
```

```
13
14      // Protoze se jedna o sestupnou hranu na lince SCL, overime
15      // priznak platnosti prijatych dat, tj. zda jiz doslo k prijetu
16      // celeho bajtu.
17      // Signalizovat prijeti bajtu bude vzdy jen jedno zarizeni.
18      if (dev->IsDataReady()) {
19        SetDeviceData(dev->GetDeviceNum(), dev->GetData());
20      }
21    } // for (dev = ...
22
23    // Pokud led SCL sviti, zhasneme ji
24    if (mLedLine[0].GetLedState()) {
25      mLedLine[0].SetLedState(false);
26      mLedLine[0].queue_draw();
27    }
28  }
```

Výpis 11: Zpracování zprávy RST_SCL

V případě, že klient žádá o zaslání zprávy o stavu výstupů linky SDA simulovaných Slave zařízení, obdrží primární vlákno příslušnou zprávu, která je zpracována způsobem uvedeným ve výpisu programu 12. Metoda opět projde všechna simulovaná zařízení, pomocí logického součinu zjistí výsledný stav linky SDA a nakonec vytvoří text zprávy pro klienta a odešle mu ho.

```
1  void MainWindow::on_send_sda() {
2    char SendBuff[BUFFER_SIZE];
3    int RetVal;
4    vector<I2C>::iterator dev;
5
6    // Projdeme vsechna Slave zarizeni a zjistime stav vystupu SDA
7    // na vsech zarizenich.
8    // Na realne lince SDA muze byt pripojeno vice Slave zarizeni a
9    // pokud jedno z nich nastavi SDA na nizkou uroven, nastavime
10   // mSDA_Out na hodnotu 0.
11   UCHAR mSDA_Out = 1;
12
13   for (dev = mDevices.begin(); dev != mDevices.end(); dev++) {
14     mSDA_Out &= dev->SDA();
15   }
16
17   sprintf_s (SendBuff, "SDA=%01x", mSDA_Out);
18   RetVal = this->SendMessage(mClientSockets[0], SendBuff);
19
20   if (RetVal)
21     this->MessageInfoBox("Zprava " + Glib::ustring(SendBuff) + " neodeslana.", Gtk::
           MESSAGE_INFO);
22 }
```

Výpis 12: Zpracování zprávy GET_SDA

Když primární vlákno obdrží od pomocného vlákna ServiceThread oznámení o připojení klienta, zobrazí ve stavovém řádku informace o klientovi (jeho názvu, IP adrese a portu) a vytvoří nové pomocné vlákno ClientThread, jehož metodě MainWindow ::ClientThread(SOCKET ClientSock) předá v parametru číslo soketu, na kterém bude probíhat komunikace s klientem (viz výpis programu 13). Po vytvoření vlákna toto vlákno ihned zahájí činnost. Pokud klient ukončí svoji činnost nebo je spojení přerušeno z jiného důvodu, je běh pomocného vlákna ClientThread ukončen.

```
1  void MainWindow::on_client_connected() {
2    char        clInfo [50];
3    ClientInfo  NewInfo;
4
5    // Udelame kopii s informacemi klienta
6    Glib :: Mutex::Lock lock(ClientChangeMutex);
7    memcpy(&NewInfo, &mClientInfo, sizeof(NewInfo));
8    lock.release();
9
10   if (NewInfo.Port != 0) {
11     sprintf_s ( clInfo , "Client : %s (%s:%d)", NewInfo.Name, NewInfo.Ip, NewInfo.Port);
12   }
13
14   mStatusBar.push(clInfo);
15   mStatusBar. set_tooltip_text ( clInfo ) ;
16
17   mClientThreadActive = true;
18
19   // Vytvorime klientske vlakno a predame mu cislo soketu pripojeneho klienta
20   mClientThread = Glib::Thread::create(
21     sigc :: bind(sigc :: mem_fun(* this, &MainWindow::ClientThread), mClientSockets[0]), false);
22 }
```

Výpis 13: Vytvoření nového vlákna po připojení klienta

Primární vlákno je samozřejmě zodpovědné za provedení obsluhy událostí vznikajících na podnět uživatele prostřednictvím GUI aplikace. Jednou z těchto událostí je změna hodnoty napětí přiváděného na vstup AD převodníku. V případě, že dojde ze strany uživatele ke změně hodnoty vstupního napětí použitím posuvníku (nebo změně referenčního napětí) je jako reakce na tuto událost provedena metoda MainWindow:: on_reg_voltage_value_changed(). Kód této metody uskuteční převod analogové hodnoty na její číselné vyjádření v rozsahu osmi bitů. Převod je proveden dle vztahu 1 uvedeném na straně 29.

```
1  void MainWindow::on_reg_voltage_value_changed() {
2    // Prevod napetove urovne na ciselne vyjadreni (A/D prevod pri 8bit kvantizaci)
3    unsigned char ConvertValue =
4      (unsigned char) ((mDev2VoltReg.get_value() * 255) / mDev2RefVoltage.get_value());
5
6    // Dalsi kod...
7
```

```
8    // Nova vstupni data ulozime do buferu zarizeni
9    mDevices[1].SetData(ConvertValue);
10 }
```

Výpis 14: AD převod

Při změně vstupních dat u zařízení expandéru PCF8574 provedených uživatelem na kterékoliv lince portu je volána metoda MainWindow::on_port_value_dev1_changed(**int** Pin, **bool** Value), která obslužnému kódu předá číslo změněné linky a její nový stav. Metoda opět po přepočtu nastaví v buferu zařízení novou vstupní hodnotu (viz zkráceny výpis programu 15).

```
1  void MainWindow::on_port_value_dev1_changed(int Pin, bool Value) {
2    // Ziskame puvodni hodnotu na portu
3    Glib :: ustring lblValue = mLblDevSendValue[0].get_text();
4    Num = atoi(lblValue.substr(0, 3). c_str () ) ;
5
6    // Hodnotu upravime podle noveho stavu
7    if (Value)
8      Num += 1 << Pin;
9    else
10     Num -= 1 << Pin;
11
12   // Dalsi kod...
13
14   // Nova vstupni data ulozime do zarizeni
15   mDevices[0].SetData(Num);
16 }
```

Výpis 15: Změna hodnoty na vstupních linkách expandéru

Tak jak již bylo uvedeno na straně 39 v podkapitole 7.1, primární vlákno rovněž zajišťuje překreslování grafických objektů v GUI aplikace. O překreslování standardních ovládacích prvků se starají tyto prvky automaticky, nicméně u grafických prvků, které jsou vykreslovány programově, je nutno se o překreslování postarat. V aplikaci jsou všechny LED diody zobrazené v okně aplikace překreslovány vždy, když je to potřeba (změna viditelnosti prvků v okně, minimalizace nebo maximalizace okna apod.) a to pomocí metody LedSymbol::on_expose_event(GdkEventExpose * Event) třídy LedSymbol. Událost on_expose_event() je systémem automaticky generována v případě nutnosti překreslení grafiky. Tato událost je v situacích, kdy dochází ke změně stavu LED diody (zhasnutá ↔ rozsvícená), vyvolána také programově. Implementace této metody je uvedena ve výpisu kódu 16.

```
1  bool LedSymbol::on_expose_event(GdkEventExpose * Event) {
2    // Ziskame kontex pro vykreslovani
3    Glib :: RefPtr<Gdk::Window> window = get_window();
4
5    if (window) {
6      Gtk:: Allocation allocation = get_allocation () ;
```

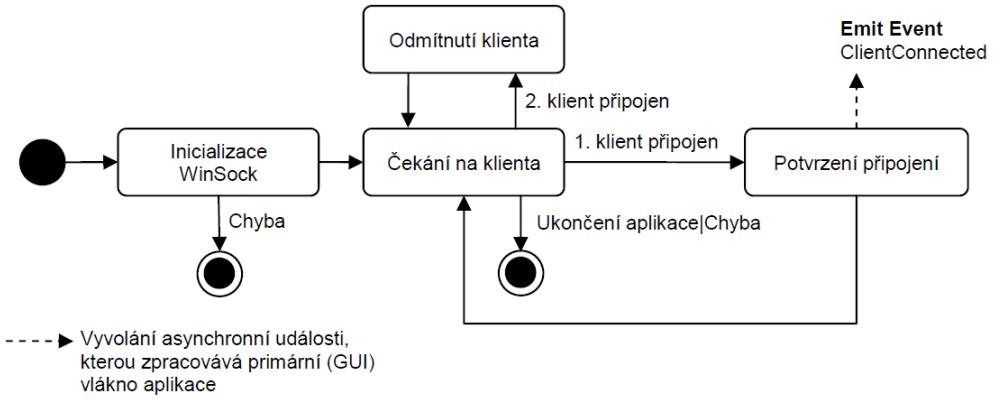
```
7       const int width = allocation.get_width();
8       const int height = allocation.get_height();
9       int lesser = MIN(width, height);
10
11      // Souradnice pro stred okna
12      int xc, yc;
13      xc = width / 2;
14      yc = height / 2;
15
16      Cairo::RefPtr<Cairo::Context> cr = window->create_cairo_context();
17
18      cr->set_line_width(lesser * 0.02); // Tloustka linky
19
20      // Upravime oblast pro preskresleni tak, ze bude prekreslovana
21      // je ta cast vyrezu, kterou je treba skutecne prekreslit
22      cr->rectangle(Event->area.x, Event->area.y,
23                    Event->area.width, Event->area.height);
24
25      cr->clip();
26
27      // Vykreslime plnou kruznici
28      cr->save();
29      cr->arc(xc, yc, lesser / 2.4, 0.0, 2.0 * PI);
30
31      // Podle aktualniho stavu diody vykreslime vnitrek diody
32      // Barevne slozky jsou nastavovany procentualne: 1.0 = 100%
33      if (this->mLedState) {
34        // Nastavime barvu svitici led
35        cr->set_source_rgb(
36          mLedColorOn.Red / 255.0, mLedColorOn.Green / 255.0, mLedColorOn.Blue / 255.0);
37      }
38      else {
39        // Nastavime barvu zhasnute led
40        cr->set_source_rgb(
41          mLedColorOff.Red / 255.0, mLedColorOff.Green / 255.0, mLedColorOff.Blue / 255.0);
42      }
43
44      cr->fill_preserve();
45      cr->restore();
46      cr->stroke();
47    }
48    return true;
49  }
```

Výpis 16: Způsob překreslování LED diod v GUI aplikace

### 7.3.2 Servisní vlákno

Servisní vlákno ServiceThread je reprezentováno metodou MainWindow::ServiceThread(). Toto vlákno je vytvořeno při inicializaci aplikace a ihned zahajuje svoji činnost. Stavový diagram vlákna ServiceThread je na obrázku 30.

Vlákno ServiceThread nejprve ověří dostupnost rozhraní `WinSock` v operačním systému a poté vytvoří na daném portu nový soket, na kterém bude aplikace naslouchat příchozím spojením. Servisní vlákno naslouchá pomocí funkce select() rozhraní `WinSock`, která zablokuje chod vlákna a čeká, dokud nenastane sledovaná událost (připojení klienta) na soketu. U funkce select() je využito možnosti definovaného časového limitu (timeoutu), po jehož uplynutí pokračuje vlákno v činnosti, čehož aplikace využívá ke zjištění, zda primární vlákno nepožaduje ukončení činnosti vlákna ServiceThread (při ukončení aplikace). V případě, že není po uplynutí časového limitu požadováno ukončení činnosti vlákna ServiceThread, pokračuje vlákno dále v čekání na sledovanou událost. Časový limit pro funkci select() je nastaven na 50 ms.




Obrázek 30: Stavový diagram vlákna ServiceThread

Po úspěšném spojení s klientem servisní vlákno vytvoří nový soket, na kterém bude probíhat oboustranná komunikace s klientem. Poté se vlákno pokusí upravit vlastnosti vytvořeného soketu nastavením atributu `TCP_NODELAY` skupiny `IPPROTO_TCP`.

Atribut `TCP_NODELAY` zablokuje na daném soketu použití Nagleova algoritmu. Nagleův algoritmus slouží ke snížení zatížení sítě. Algoritmus spočívá v tom, že data předávána jednotlivými voláními k vyslání na virtuální kanál nejsou vysílána okamžitě, ale jsou shromažďována v buferu a vyslána až v okamžiku, kdy je shromážděno takové množství dat, které zaplní TCP segment s určitou délkou (nebo uplyne časový limit). Tím se omezí posílání krátkých paketů, které by jednak měly nevyhovující poměr významové informace k informaci servisní (hlavičce) a jednak by zcela zbytečně vyžadovaly potvrzování každého krátkého segmentu samostatně. Takto by celkový objem režijních dat přenášených sítí vzrostl. Vzhledem k charakteru zpráv, které klient serveru posílá (krátké a časté zprávy simulující změny logických úrovní), přinese blokování Nagleova algoritmu zvýšení přenosové rychlosti simulované sběrnice. Pokud však bude aplikace simulátoru a klienta pracovat na dvou nezávislých počítačích ve skutečném síťovém prostředí se silným provozem, efekt zvýšení rychlosti nebude zřejmě příliš významný.

Pokud byl k simulátoru připojen první klient, servisní vlákno získá informace o připojeném klientovi, které uloží do sdíleného atributu mClientInfo třídy MainWindow, zašle

klientovi zprávu `CONNECTED!OK...` a poté zašle primárnímu vláknu oznámení o připojení klienta. Primární vlákno na základě tohoto oznámení vytvoří nové pomocné vlákno ClientThread, které bude s klientem komunikovat (viz částečný výpis programu 17). V případě, že byl k simulátoru připojen druhý klient, servisní vlákno zašle tomuto klientovi pouze zprávu `DECLINED.SORRY.` a spojení zruší. Servisní vlákno pak pokračuje v naslouchání.

```
1  if (ClientCount == 1) {
2    // Prvni klient pripojen
3    // Odesleme pripojenemu klientovi potvrzeni o pripojeni
4    strcpy_s(SendBuffer, MSG_CONNECTED);
5    send(mClientSockets[ClientCount - 1], SendBuffer, (int) strlen(SendBuffer) + 1, 0);
6
7    // Zapiseme do sdileneho atributu informace o klientovi
8    Glib::Mutex::Lock lock(ClientChangeMutex);
9    memcpy(&this->mClientInfo, &clInfo, sizeof(clInfo));
10   lock.release();
11
12   // Posleme signal GUI vlaknu o pripojeni klienta
13   mClientConnected.emit();
14 }
15 else
16   // ...
```

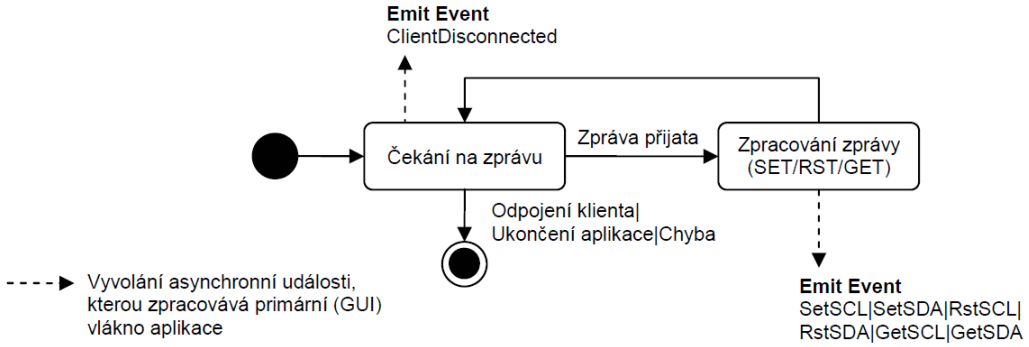
Výpis 17: Operace po připojení prvního klienta

### 7.3.3 Klientské vlákno

Klientské vlákno ClientThread je reprezentováno metodou MainWindow::ClientThread(). Toto vlákno je vytvořeno po připojení klienta a okamžitě čeká na jeho zprávu. Čekání na příchozí zprávu je prováděno pomocí blokující funkce recv() rozhraní `WinSock`. Po přijetí zprávy vlákno pokračuje v činnosti a zprávu uloží do strukturované proměnné typu ClientMsg, kterou předá metodě MessageProcess() ke zpracování. Po zpracování zprávy vlákno opět čeká na další zprávu. Stavový diagram vlákna ClientThread je na obrázku 31.

Metoda MessageProcess() podle typu zprávy zašle příslušné oznámení primárnímu vláknu, které toto oznámení zpracuje. Způsob implementace metody je uveden ve výpisu programu 18.

```
1  void MainWindow::MessageProcess(ClientMsg * Msg) {
2    if (!strcmp(Msg->Command, "SET")) {
3      // Zpracovani zpravy SET...
4      if (!strcmp(Msg->Line, "SCL")) {
5        mSetSCL.emit();
6      } else if (!strcmp(Msg->Line, "SDA")) {
7        mSetSDA.emit();
8      }
9    } // SET
10
```




Obrázek 31: Stavový diagram vlákna ClientThread

```
11    else if (!strcmp(Msg->Command, "RST")) {
12      // Pozadavek na SCL=0
13      if (!strcmp(Msg->Line, "SCL")) {
14        mRstSCL.emit();
15      }
16      else if (!strcmp(Msg->Line, "SDA")) {
17        mRstSDA.emit();
18      }
19    } // RST
20
21    else if (!strcmp(Msg->Command, "GET")) {
22      // Zpracovani GET... (Master pozaduje stav linky)
23      if (!strcmp(Msg->Line, "SCL")) {
24        // Master pozaduje stav linky SCL
25        mGetSCL.emit();
26      }
27      else if (!strcmp(Msg->Line, "SDA")) {
28        // Master pozaduje stav linky SDA
29        mGetSDA.emit();
30      }
31    } // GET
32  }
```

Výpis 18: Zpracování zprávy klienta

V případě, že klient z nějakého důvodu ukončí se simulátorem spojení, pomocné vlákno ClientThread zašle primárnímu vláknu oznámení o této skutečnosti a poté taktéž ukončí svoji činnost.

## 7.4 Ukončení aplikace

Při ukončení aplikace dochází nejprve k ukončení činnosti obou pomocných vláken ServiceThread a ClientThread (pokud existuje). V případě, že je uživatelem nastaveno, aby aplikace při ukončení uložila nastavení GUI prostředí, je navíc zavolána metoda

SaveConfiguration() třídy Settings a ta zajistí uložení všech uživatelských nastavení do konfiguračního souboru aplikace.

Bližší informace o konfiguračních nastaveních jsou popsány v podkapitole 7.5.

## 7.5 Nastavení aplikace

Aplikace simulátoru využívá k uchování uživatelských nastavení v prostředí GUI samostatný soubor s názvem `i2c_server.ini`, který je umístěn ve stejné složce jako aplikace. V případě, že aplikace při spuštění soubor nenalezne, použije standardní nastavení.

Pro implementaci uživatelských nastavení bylo využito třídy Glib::KeyFile, která poskytuje vše potřebné pro práci s konfiguračními soubory. Třída poskytuje metody pro čtení a pro zápis do konfiguračního souboru, práci s různými typy dat apod. S použitím této třídou byla vytvořena třída Settings, která poskytuje rozhraní pro třídu MainWindow okna aplikace. Tato třída poskytuje metody pro načtení nastavení ze souboru (metoda LoadConfiguration()), zápis do souboru (metoda SaveConfiguration()) a metodu pro inicializaci standardních nastavení LoadDefaultSetting() v případě neexistence konfiguračního souboru.

Konfigurační soubor `i2c_server.ini` obsahuje jednotlivá nastavení v pojmenovaných sekcích ve formátu `klíč=hodnota`. Popis jednotlivých nastavení je uveden v tabulkách 9 a 10.

Ukázka obsahu konfiguračního souboru je uvedena v příloze D.

| **Sekce** | **Klíč** | **Výchozí hodnota** | **Význam** |
|---|---|---|---|
| Server | Port | 3000 | Číslo portu, na kterém aplikace naslouchá požadavkům na připojení. Musí být zadána celočíselná nezáporná hodnota v rozsahu 1024 až 65535. |
| Device1 | Address | 64 | Adresa Slave zařízení číslo 1 (expandéru PCF8574). Adresu je nutné zapsat v decimálním vyjádření. |
| | Visible | true | Hodnota `true\|false` nastavuje, zda rám se zařízením bude viditelný či nikoli. |
| | VisibleState | true | Hodnota `true\|false` nastavuje, zda bude v rámu zařízení zobrazen jeho vnitřní stav či nikoli. |
| | VisibleSend | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude v rámu zařízení zobrazena hodnota nastavená na vstupních linkách zařízení. |
| | VisibleReceive | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude v rámu zařízení zobrazena hodnota nastavená na výstupu zařízení. |
| Device2 | Address | 96 | Adresa Slave zařízení číslo 2 (AD převodníku). Adresu je nutné zapsat v decimálním vyjádření. |
| | Visible | true | Hodnota `true\|false` nastavuje, zda rám se zařízením bude viditelný či nikoli. |
| | VisibleState | true | Hodnota `true\|false` nastavuje, zda bude v rámu zařízení zobrazen jeho vnitřní stav či nikoli. |
| | VisibleSend | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude v rámu zařízení zobrazena hodnota s výsledkem AD převodu. |
| | VisibleReceive | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude v rámu zařízení zobrazena hodnota přijatá od zařízení Master. |

Tabulka 9: Popis konfiguračních voleb – sekce Server, Device1 a Device2

| Sekce | Klíč | Výchozí hodnota | Význam |
|---|---|---|---|
| Window | AlwaysOnTop | false | Hodnota `true\|false` nastavuje, zda bude okno aplikace vždy nad ostatními okny či nikoli. |
| | SaveOnExit | true | Hodnota `true\|false` nastavuje, zda má aplikace při ukončení uložit uživatelská nastavení do konfiguračního souboru či nikoli. |
| | ViewControlStatus | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude zobrazen rám s indikačními LED diodami linek SCL a SDA a tlačítky pro úpravu adresy. |
| | ViewStatusBar | true | Hodnota `true\|false` nastavuje, zda bude nebo nebude zobrazen stavový řádek aplikace. |
| | Position | 0;0 | Souřadnice X a Y pozice okna v pixelech. |
| | LedOnRGB | 255;0;0 | Barva LED diody v rozsvíceném stavu ve formátu RGB. Výchozí hodnotou je červená. |
| | LedOffRGB | 128;0;0 | Barva LED diody ve zhasnutém stavu ve formátu RGB. Výchozí hodnotou je tmavě červená. |

Tabulka 10: Popis konfiguračních voleb – sekce Window

# 8 Porovnání chování se skutečnými obvody

## 8.1 Expandér PCF8574

Funkce integrovaného obvodu PCF8574 jsou simulovaným zařízením plně implementovány, kromě simulace výstupu $\overline{\mathbf{INT}}$ (INTerrupt). Tento výstup nebyl v simulovaném zařízení implementován, neboť komunikace mezi zařízením Master (klientem) a zařízením Slave (simulátorem) měla probíhat výhradně prostřednictvím signálů simulujících $I^2C$ sběrnici bez dalších signálních linek. Vytvořený komunikační protokol pro přenos dat mezi zařízením Master a Slave proto nezahrnuje zprávy týkající se jiných signálu než SCL a SDA.

Drobnou úpravou v programu simulátoru by sice bylo možné doplnit indikaci stavu výstupu $\overline{\mathbf{INT}}$ skutečného obvodu, tato indikace by však neměla pro uvažované využití simulátoru velkého významu.

## 8.2 Analogový převodník

Vzhledem k tomu, že v zadání práce nebyl specifikován typ skutečného obvodu AD převodníku, byl pro účely simulace vytvořen pouze fiktivní AD převodník, který by měl vlastnosti a chování obdobné skutečnému převodníku.

Převod je iniciován uživatelem vždy při změně úrovně napětí na vstupu převodníku. Převod je proveden v konečné době a ihned zapsán do vnitřního buferu zařízení. Zařízení Master má při čtení hodnoty ze zařízení AD převodníku hodnotu převodu odpovídající aktuálně nastavenému napětí na vstupu převodníku.

Pozitivní odlišností oproti skutečnému AD převodníku je, že u simulovaného AD převodníku není převod negativně ovlivňován technickými parametry obvodu, např. nelinearita převodu, vlastní šum, rušení, stabilita referenční napětí apod.

# 9 Testování spolehlivosti a zhodnocení

K testování spolehlivosti simulátoru byly vytvořeny na klientské straně tři testovací funkce, jejichž zdrojový kód je uveden ve výpisu kódu 19. Při testování byla zjišťována stabilita spojení klienta se simulátorem v různých provozních situacích, např. uměle vyvolané zatížení počítače, minimalizace a přesun okna, manipulace s GUI rozhraním simulátoru, připojení druhého klienta, zobrazení dialogu s údaji o aplikaci apod. Během aktivního provozu simulátoru byla také ověřována rychlost a správnost na podněty vyvolané uživatelem v GUI simulátoru.

Testování bylo prováděno jak v prostředí, kdy klient a simulátor spolu komunikoval v rámci jednoho počítače a tak v prostředí domácí bezdrátové sítě, kdy klient pracoval na přenosném počítači a simulátor na stolním počítači.

Seznam testovacích funkcí:

1. Opakované čtení hodnoty AD převodu s periodou 100 ms (I2C_ReadDataFromAD()).
2. Opakující zápis bloku 32 bajtů na výstupní port expandéru PCF8574 imitující různé světelné efekty s periodou 150 ms (I2C_WriteDataToExp()).
3. Opakovaný zápis čtyř bajtů na výstupní port expandéru PCF8574 následovaný čtením hodnoty na vstupu portu s periodou 150 ms. (I2C_ReadWriteDataToExp()).

Aplikace simulátoru byla testována na dvou počítačích (stolním a přenosném), jejichž konfigurace je popsána níže.

**1. Stolní počítač**

- Procesor Intel®Pentium®Dual CPU E2140 1,60 GHz
- Operační paměť RAM 2 GB
- Operační systém Microsoft Windows XP Professional (Service Pack 3)

**2. Přenosný počítač Acer Aspire 1830T**

- Procesor Intel®Core™i3 CPU U380 1,33 GHz
- Operační paměť RAM 4 GB
- Operační systém Microsoft Windows 7 Home Premium

```
1 void I2C_ReadDataFromAD(UCHAR SlaveAddress, int Cycles) {
2   UCHAR Data = 0;
3   const double Vref = 5.00;
4
5   i2c_SendSTART();
```

```
6
7    i2c_SendData(SlaveAddress + FLAG_R);
8
9    // Zjistime stav ACK
10   if (i2c_GetAck() == 1) {
11     // Adresa neprijata -> STOP
12     i2c_SendSTOP();
13     return;
14   }
15
16   while (Cycles--) {
17     Data = i2c_ReceiveData();
18      printf ("M: Prijata data: %d [%1.2f V]\n", Data, (Data * Vref) / 255);
19
20     i2c_SendAck();
21
22     Sleep(100);
23   }
24
25   i2c_SendSTOP();
26
27   return;
28 }
29
30 void I2C_WriteDataToExp(UCHAR SlaveAddress, int Cycles) {
31   UCHAR Buffer[32] = {
32     1, 2, 4, 8, 16, 32, 64, 128,
33     128, 64, 32, 16, 8, 4, 2, 1,
34     129, 195, 231, 255, 126, 60, 24, 0,
35     170, 84, 170, 84, 170, 84, 170, 84
36   };
37
38   i2c_SendSTART();
39
40   i2c_SendData(SlaveAddress + FLAG_W);
41
42   if (i2c_GetAck() == 1) {
43     // Adresa neprijata -> STOP
44     i2c_SendSTOP();
45     return;
46   }
47
48   while (Cycles--) {
49     for (int dt = 0; dt < 32; dt++) {
50       i2c_SendData(Buffer[dt]);
51        printf ("M: Odeslana data: %d\n", Buffer[dt]);
52
53       if (i2c_GetAck() == 1) {
54         i2c_SendSTOP();
55         return;
56       }
57       Sleep(150);
58     }
59   }
```

```
60
61    i2c_SendSTOP();
62
63    return;
64  }
65
66  void I2C_ReadWriteDataToExp(UCHAR SlaveAddress, int Cycles) {
67    UCHAR Data = 0;
68    UCHAR Buffer[4] = {129, 24, 65, 36};
69
70    while (Cycles--) {
71      // Zahajime komunikaci
72      i2c_SendSTART();
73
74      // Zasleme adresu + Write
75      i2c_SendData(SlaveAddress + FLAG_W);
76
77      // Zjistime stav ACK
78      if (i2c_GetAck() == 1) {
79        // Adresa neprijata -> STOP
80        i2c_SendSTOP();
81        return;
82      }
83
84      // Odesleme zarizeni Slave 4 bajty
85      for (int dt = 0; dt < 4; dt++) {
86        i2c_SendData(Buffer[dt]);
87        printf ("M: Odeslana data: %d\n", Buffer[dt]);
88
89        // Zjistime, co na odeslana data rika Slave
90        if (i2c_GetAck() == 1) {
91          // Data neprijata nebo slave ukoncuje prijem -> STOP
92          i2c_SendSTOP();
93          return;
94        }
95        Sleep(150);
96      }
97
98      // Repeated START (prepneme na cteni)
99      i2c_SendSTART();
100
101     i2c_SendData(SlaveAddress + FLAG_R);
102
103     if (i2c_GetAck() == 1) {
104       // Adresa neprijata -> STOP
105       i2c_SendSTOP();
106       return;
107     }
108
109     // Cteni dat ze slave
110     Data = i2c_ReceiveData();
111     printf ("M: Prijata data: %d\n", Data);
112
113     // Potvrdime prijem
```

```
114       i2c_SendAck();
115
116       Sleep(150);
117     }
118
119     // Ukoncime komunikaci
120     i2c_SendSTOP();
121
122     return;
123   }
```

Výpis 19: Testovací funkce klienta

Během testování stability a funkcionality v různých provozních podmínkách simulátoru nebyly zjištěny žádné problémy. Při aktivní komunikaci simulátoru s klientem bylo pouze zjištěno, že dochází ke zpomalení funkce simulátoru při současném přesouvání okna simulátoru pomocí myši. Hlavní smyčka zpráv okna simulátoru je nucena v případě přesouvání okna zpracovávat větší množství systémových zpráv a z tohoto důvodu logicky dochází také ke zpomalení obsluhy zpráv, které jsou generovány programově pomocným vláknem ClientThread. Všechny události o došlých zprávách ze strany klienta jsou ukládány do fronty zpráv, při zpomalení činnosti simulátoru proto nedochází k jeho nestabilitě či nějakým jiným problémům.

Při testování spojení klienta a simulátoru odděleně na dvou počítačích v domácí síti nebyly rovněž zjištěny problémy se stabilitou komunikace. Komunikace byla uskutečňována prostřednictvím domácího bezdrátového routeru TP-LINK TL–WR542G (WiFi 802.11 b/g, rychlost až 54 Mbit/s). Při přenosu byla uměle na komunikačním kanálu zvětšována zátěž jiným síťovým provozem využívajícím protokoly HTTP a SMTP. Rychlost předávání zpráv mezi klientem a simulátorem je závislá na aktuálním vytížení komunikačního kanálu, v případě jeho velkého zatížení se projeví zpomalením a zpožděním reakcí simulátoru avšak nijak neovlivňuje stabilitu jeho chodu.

Při souběžném aktivním provozu simulátoru a klienta na jednom počítači (č. 2) se podle správce úloh operačního systému pohybovalo zatížení CPU kolem hodnoty 4%, v případě čekání na klienta nebo při neaktivním provozu bylo zatížení nulové. Aplikace simulátoru alokovala při své činnosti cca 9 MB operační paměti. V rámci testování byla každá testovací funkce klienta provedena v 1000 cyklech na každém z testovacích počítačů.

Při samostatném aktivním provozu simulátoru na počítači č. 1 se zatížení CPU pohybovalo okolo 3%.

Maximální rychlost předávání zpráv (úrovní na linkách SCL a SDA) na simulované sběrnici $I^2C$ mezi klientem a serverem vysoce překračuje možnosti vizuálního odečtu stavu úrovní na výstupních linkách expandéru, které zobrazují LED diody. Ve většině případů bude nutno na straně klientské aplikace vkládat mezi jednotlivé datové rámce čekací cykly. Přesto je nutno počítat s tím, že rychlost přenosu zpráv na simulované sběrnici, přesněji hodinového taktu na lince SCL, je o několik řádů nižší oproti reálné sběrnici $I^2C$ a nemůže dosahovat frekvence ve standardním módu, tj. 100 kHz. Jedna perioda hodinového signálu SCL trvá při takové frekvenci pouze 10 $\mu$s a je nereálné takové rychlosti při výměně zpráv prostřednictvím soketů dosáhnout.

# 10 Závěr

Cílem této práce bylo vytvořit $I^2C$ Simulátor s grafickým rozhraním, který je schopen plně simulovat dvě zařízení. Dle stanovených požadavků byla vyvinuta aplikace pro prostředí operačního systému Windows, která je plně funkční a je připravena k plnohodnotnému využití. Vytvořený simulátor umožňuje zcela nahradit fyzické součástky a plně dostačuje pro zamýšlený účel využití. Oproti běžně užívaným způsobům připojení k rozhraním počítače, např. prostřednictvím linek paralelního portu nebo sériového portu RS-232 či USB, přináší použití simulátoru výhodu rychlého nasazení pro výukové nebo vývojové účely, úsporu nákladů a také vylučuje poškození obvodů vlivem nesprávné manipulace. Při použití rozhraní RS-232 a USB je navíc nutný příslušný převodník daného rozhraní na sběrnici $I^2C$. Rychlost komunikace na simulované sběrnici $I^2C$ je s velkou rezervou dostatečná pro potřebu vizuálního odečtu stavu linek výstupního portu expandéru (při zápisu na zařízení Slave).

Simulátor se dá relativně snadno rozšířit i na jiné vhodné druhy zařízení (obvodů) pracujících na sběrnici $I^2C$ v režimu Slave. Základem je použití univerzální třídy I2C pro toto zařízení, vytvoření grafického rozhraní simulujícího funkci příslušného obvodu a vložení objektu tohoto zařízení do pole tvořícího pomyslnou sběrnici $I^2C$.

Možným vylepšením aplikace může být doplnění modulu pro grafické zobrazování logických stavů na výstupech simulovaných zařízení v závislosti na čase (funkce jednoduchého logického analyzátoru) nebo ukládání výstupních hodnot do externího souboru, automatické nastavování vstupních hodnot s definovanou periodou apod. Simulátor může také plnit funkci univerzálního převodníku rozhraní $I^2C$ na vstupně–výstupní paralelní rozhraní.

I když byla aplikace simulátoru vyvinuta pro platformu Windows, je možná její úprava i pro platformu Linux/Unix. Potřebné úpravy aplikace by spočívaly ve změně hlavičkových souborů pro práci se sokety a úpravách ve volání funkcí pro práci se sokety a síťovou komunikací. V systému Linux/Unix jsou s malými rozdíly používány funkce pro zápis a čtení ze soketu (funkce read() a write() namísto send() a recv()) taktéž se liší zpracování návratových hodnot funkcí pracujících se sokety. Potřebné úpravy aplikace je možné řešit pomocí direktiv pro podmíněný překlad.

V příloze E této práce je CD nosič, na kterém je uloženo samotné programové řešení klienta i simulátoru včetně zdrojových kódů (pro Microsoft Visual C++ 2010) a instalační balík knihovny `GTKmm` pro platformu `Win32` obsahující všechny potřebné hlavičkové soubory, knihovny a dokumentaci. Na nosiči je rovněž uložena distribuční (release) verze aplikace simulátoru obsahující všechny potřebné knihovny nutné pro správný běh aplikace.

Reference, které nebyly v textu práce citovány, byly použity jako zdroj informací pro samotný vývoj aplikace simulátoru. Jedná se především o [6], [7] s informacemi o použití knihovny GTKmm a [4], [5] s problematikou rozhraní `WinSock` pro práci se sokety a síťovou komunikací. Dále byly podpůrně využity informace dle [8] týkající se rozhraní $I^2C$ při výuce v předmětu APPS (Architektury počítačů a paralelních systémů).

Patrik Kortiš

# 11 Reference

[1] NXP Semiconductors. *$I^2C$-bus specification and user manual* [online]. Rev. 5, 2012-09-09 [cit. 2014-04-07]. Dostupné z: http://www.nxp.com/documents/user_manual/UM10204.pdf

[2] Philips Semiconductors. *Product Specification, Data Sheet of PCF8574 Remote 8-bit I/O expander for $I^2C$-bus* [online]. 2002-11-22 [cit. 2014-04-07]. Dostupné z: http://www.nxp.com/documents/data_sheet/PCF8574.pdf

[3] FŮS, Michal. *Simulace LED řízených PWM jako aplikace v GUI* [online]. Ostrava, 2013 [cit. 2014-04-05]. Dostupné z: https://dspace.vsb.cz/handle/10084/98623. Diplomová práce. Vysoká škola báňská, Technická univerzita Ostrava. Vedoucí práce Ing. Petr Olivka.

[4] POLÁK, Daniel. *Programování síťových aplikací* [online]. Brno, 2006. Dostupné z: http://is.muni.cz/th/51885/fi_b/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Luděk Matyska.

[5] DOSTÁL, Radim. *Seriál Sokety a C/C++* [online]. 2003-06-02. Dostupné z: http://www.root.cz/serialy/sokety-a-cc

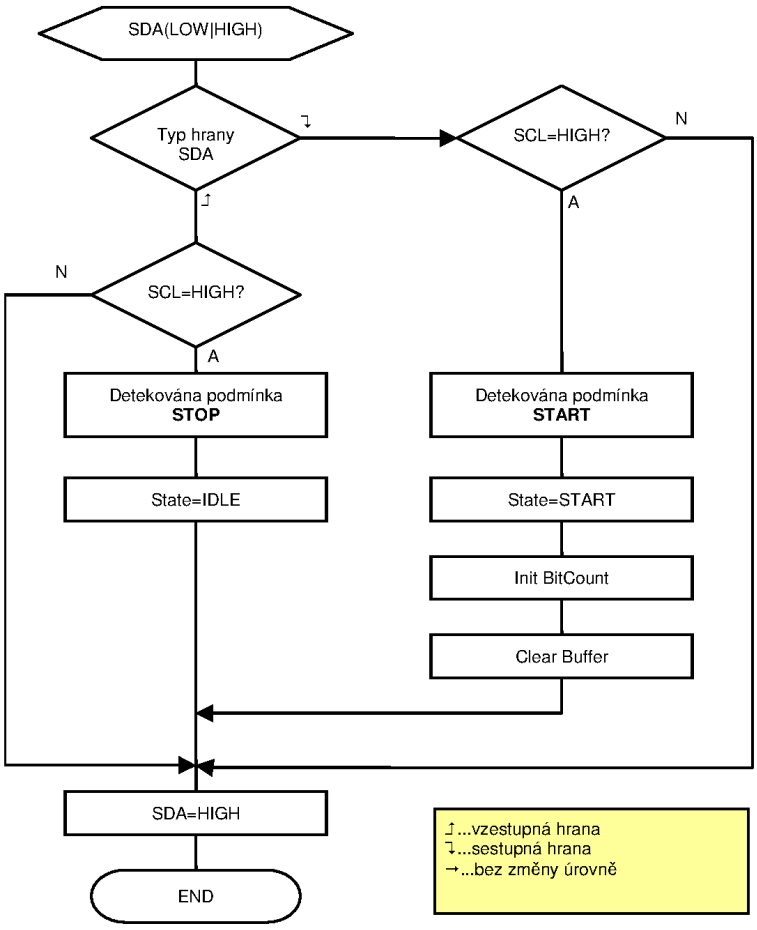
[6] CUMMING, M., B. RIEDER, J. JONGSMA, J. M'SADOQUES, O. LAURSEN, G. RUEBSAMEN, C. GUSTIN, M. ANASTASOV a A. OTT. *Programing with gtkmm* [online]. USA: Free Software Foundation, Inc., ©2002-2006. Dostupné z: http://www.murrayc.com/temp/programming-with-gtkmm.pdf

[7] RAKIC, Goran a Frederic PETERS. *Dokumentace ke gtkmm* [online]. Verze 2.24.4. Dokumentační projekt GNOME, 2008. Dostupné z: https://developer.gnome.org/gtkmm/

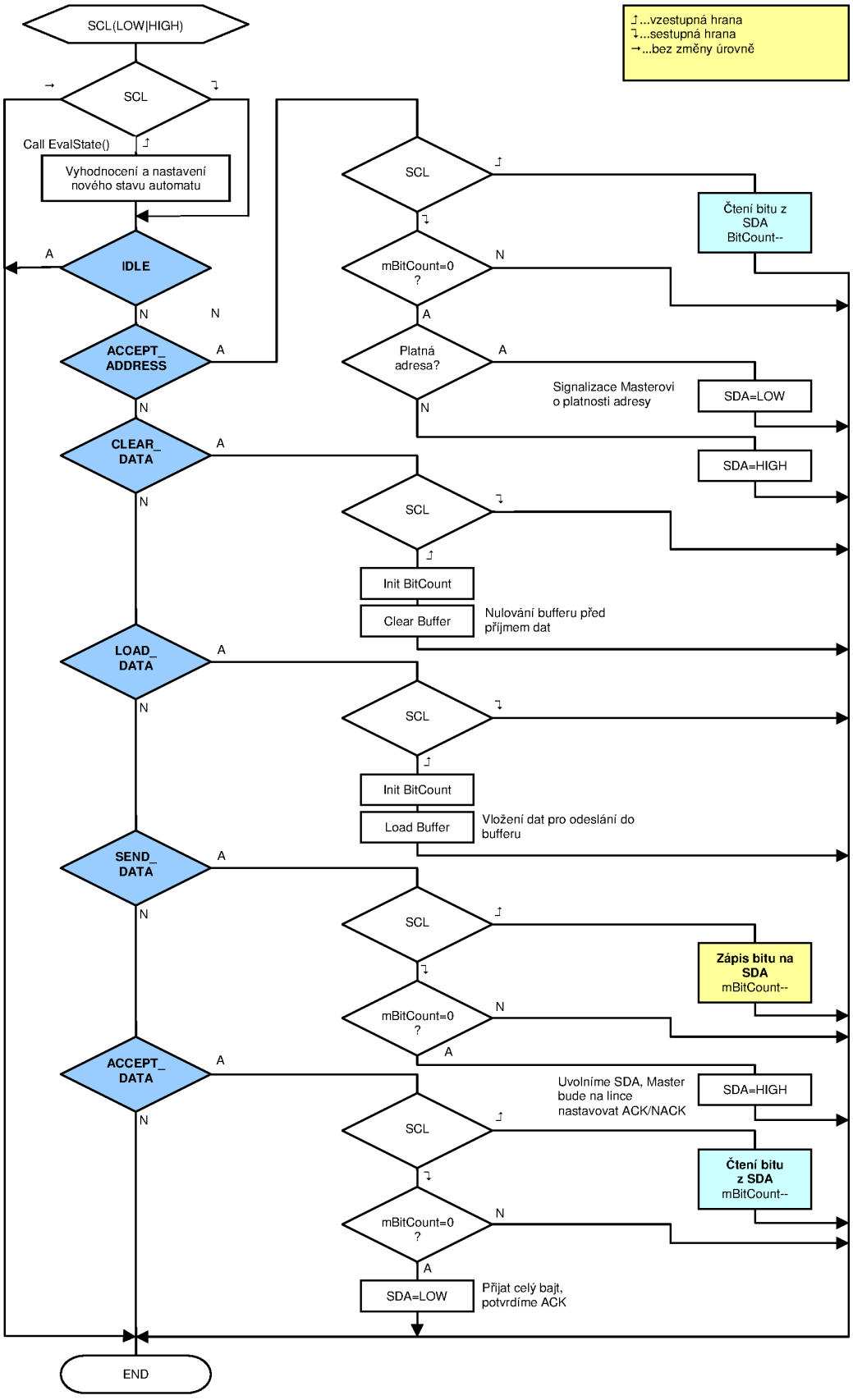
[8] OLIVKA, Petr. *Návody do cvičení předmětu APPS* [online]. 2012-18-09. Dostupné z: http://poli.cs.vsb.cz/edu/apps/lab/apps-cvic.pdf

# A Vývojové diagramy

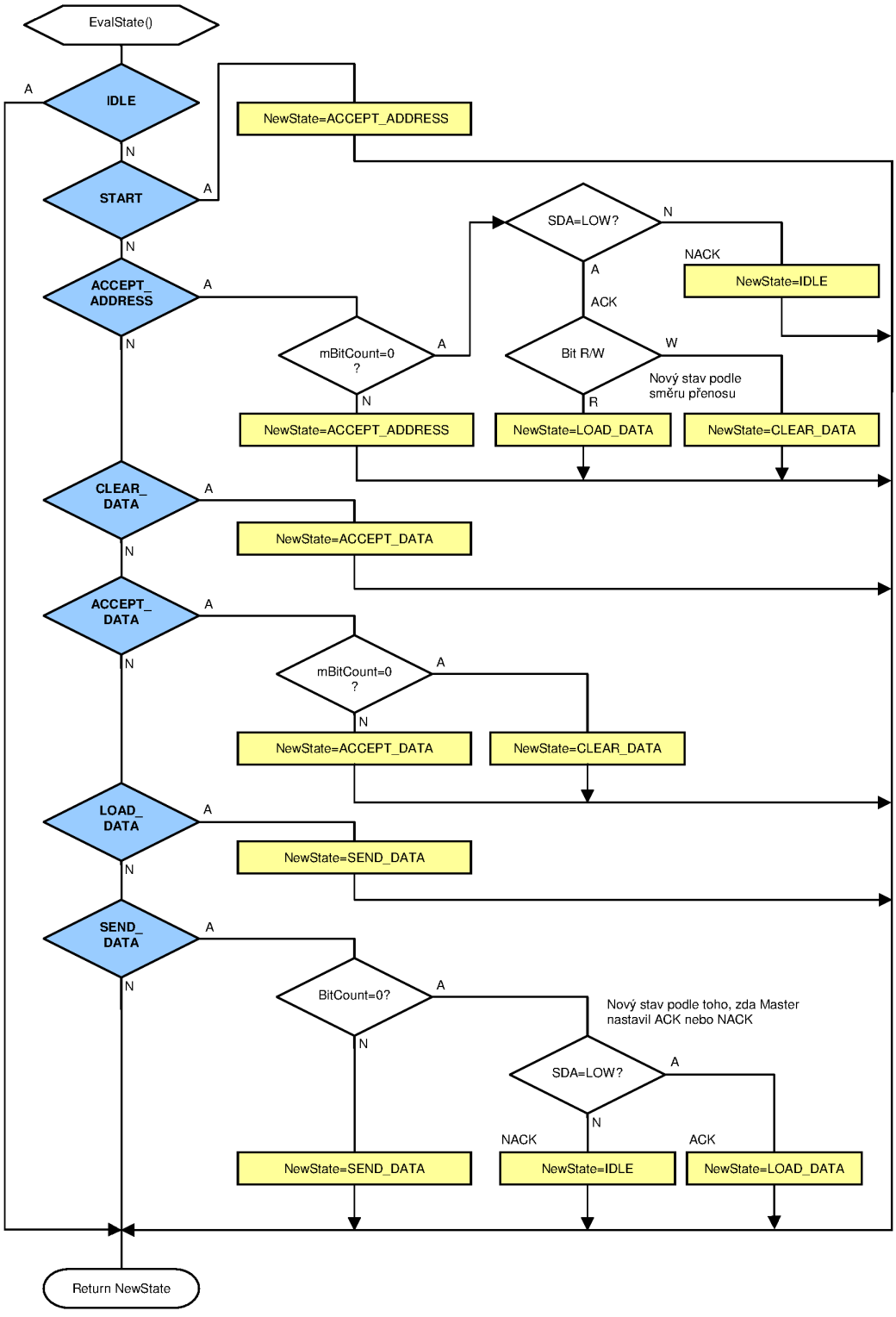
Vývojové diagramy pro metody I2C::SDA(), I2C::SCL() a I2C::EvalState().




Obrázek 32: Vývojový diagram metody SDA()



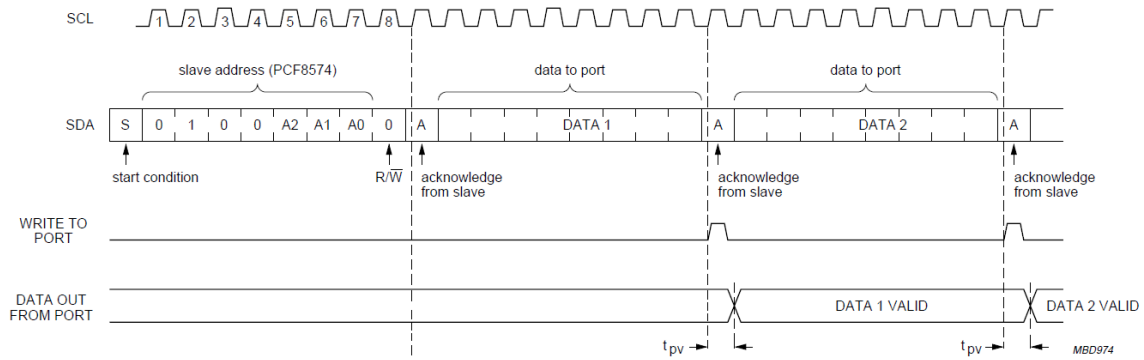

Obrázek 33: Vývojový diagram metody SCL()



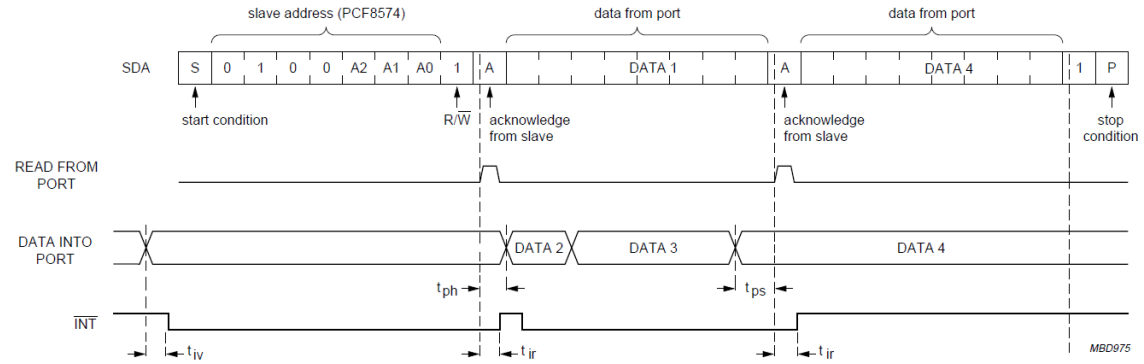

Obrázek 34: Vývojový diagram metody EvalState()

# B Časové diagramy

Časové diagramy pro zápis/čtení dat expandéru PCF8574.

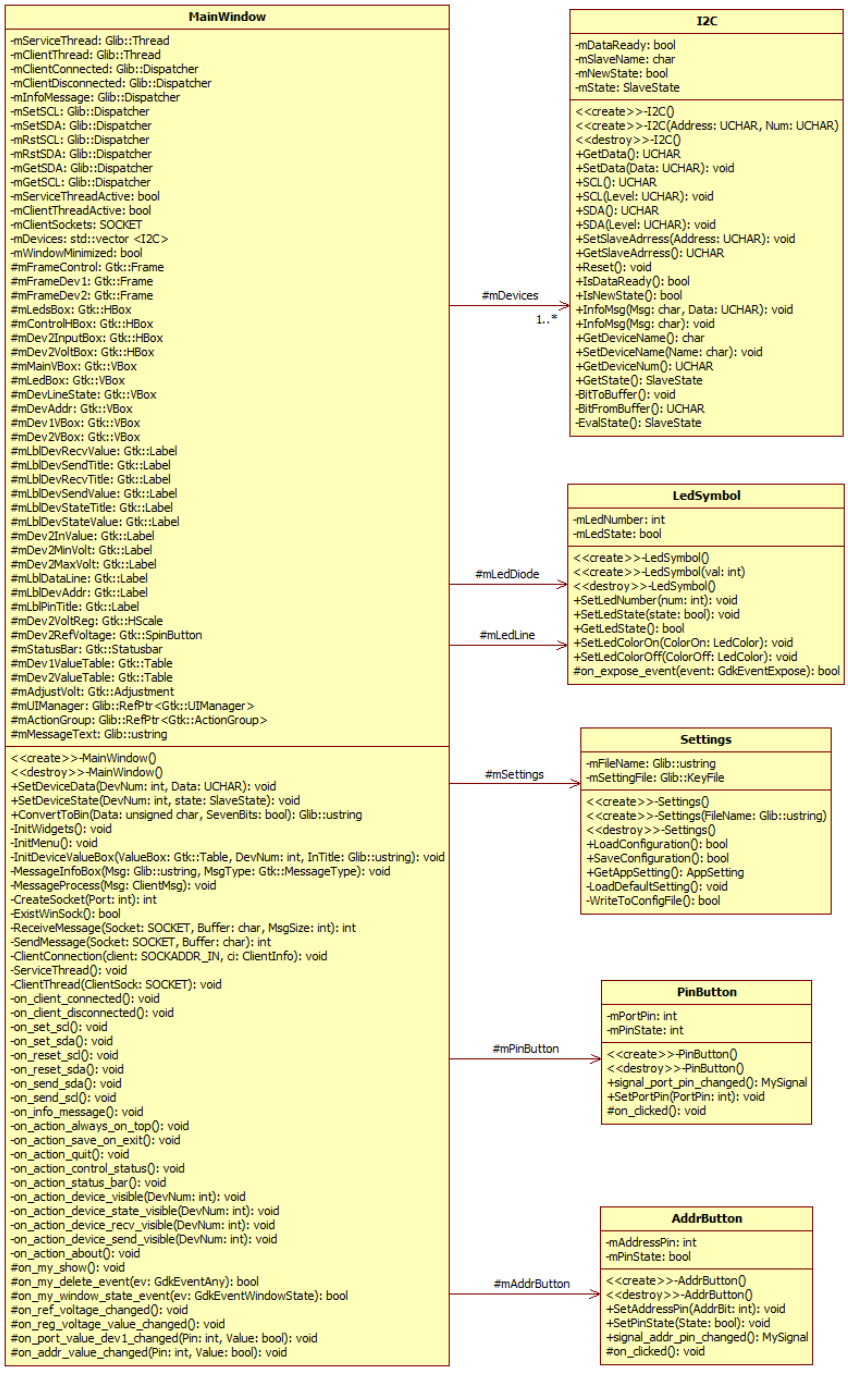



Obrázek 35: PCF8574 – zápis na port (převzato z [2])




Obrázek 36: PCF8574 – čtení z portu (převzato z [2])

# C Třídní diagramy




Obrázek 37: Třídní diagram aplikace

# D Konfigurační soubor

Ukázka konfiguračního souboru aplikace.

```
[Server]
# Cislo portu, na kterem aplikace nasloucha
Port=3000

[Device1]
# Adresa slave zarizeni c. 1 (decimalne)
Address=72
Visible=true
VisibleState=true
VisibleSend=true
VisibleReceive=true

[Device2]
# Adresa slave zarizeni c. 2 (decimalne)
Address=104
Visible=true
VisibleState=true
VisibleSend=true
VisibleReceive=true

[Window]
AlwaysOnTop=false
SaveOnExit=true
ViewControlStatus=true
LedOnRGB=255;0;0
LedOffRGB=128;0;0
ViewStatusBar=true
Position=786;120
```

# E Obsah nosiče CD